2007-12

A study to model human behavior in Discrete Event Simulation (DES) using Simkit

Tan, Boon Leng (Ryan)

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A STUDY TO MODEL HUMAN BEHAVIOR IN DISCRETE EVENT SIMULATION (DES) USING SIMKIT

by

Boon Leng (Ryan), Tan

December 2007

Thesis Advisor:	Arnold H. Buss
Second Reader:	Christian J. Darken

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE A Study to Model Human Behavior in Discrete Event Simulation (DES) using Simkit

5. FUNDING NUMBERS

6. AUTHOR(S) Boon Leng (Ryan) Tan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

In modern simulation systems, there are two main domains. The first is the Event Driven domain that uses events as the driver in the simulation. Event driven simulations are efficient as they avoid unnecessary time steps and, when carefully designed, can realistically represent real world events. The Time Driven paradigm, the other simulation domain, is favored by agent-based simulations due to the time stepping characteristic suitable for the agent decision cycle. Agent-based simulations such as the multi-agent systems are capable of modeling complex human intelligence and behavior.

In this thesis, Discrete Event Multi Agent Simulation (DEMAS) is introduced as a new design concept which provides the mean to have the best of both simulation domains. DEMAS design concept uses event graph methodology and LEGO framework to achieve design simplicity and modularity, allowing multi-agent systems to be added into a discrete event world. To validate the new design concept, three simulations of different complexity levels were developed using Simkit Java package. The validations eventually proved the worthiness of the DEMAS design concept for providing the means to build simulations with benefiting characteristics of both multi-agent systems and discrete event simulations.

14. SUBJECT TERMS Simulation, Simulation Design, Discrete Event Simulation, DES, Multi Agent Simulation, MAS, Agent Based Simulation, ABS, Event Graph, Simkit, Discrete Event Multi Agent Simulation, DEMAS, Riverine Sustainment 2012, SEA11.

15. NUMBER OF PAGES
103

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A STUDY TO MODEL HUMAN BEHAVIOR IN DISCRETE EVENT SIMULATION (DES) USING SIMKIT

Boon Leng (Ryan), Tan
Major, Republic Of Singapore Navy
B. (Hons) Comp Science, University of Melbourne, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING, VIRTUAL ENVIRONMENTS AND SIMULATION (MOVES)

from the

NAVAL POSTGRADUATE SCHOOL
December 2007

Author: Boon Leng (Ryan), Tan

Approved by: Professor Arnold H. Buss
Thesis Advisor

Professor Christian J. Darken
Second Reader

Professor Rudolph P. Darken
Chair, Department of MOVES Institute

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In modern simulation systems, there are two main domains. The first is the Event Driven domain that uses events as the driver in the simulation. Event-driven simulations are efficient as they avoid unnecessary time steps and, when carefully designed, can realistically represent real-world events. The Time Driven paradigm, the other simulation domain, is favored by agent-based simulations due to the time stepping characteristic suitable for the agent decision cycle. Agent-based simulations such as the multi-agent systems are capable of modeling complex human intelligence and behavior.

In this thesis, Discrete Event Multi Agent Simulation (DEMAS) is introduced as a new design concept that provides the means to have the best of both simulation domains. DEMAS design concept uses event graph methodology and LEGO framework to achieve design simplicity and modularity, allowing multi-agent systems to be added into a discrete event world. To validate the new design concept, three simulations of different complexity levels were developed using the Simkit Java package. The validations eventually proved the worthiness of the DEMAS design concept for providing the means to build simulations with beneficial characteristics of both multi-agent systems and discrete event simulations.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

My most sincere gratitude and appreciation goes to my advisor, Professor Arnold H. Buss, from whom I have learnt much about Discrete Event Simulation and Simkit. I gratefully acknowledge his patient support and guidance, which has made this thesis possible. It is a privilege to have worked with him for the past one year. His knowledge and expertise have truly inspired me in many ways.

I would like to thank Professor John Hiles for allowing me to reference some of his teaching materials for multi-agent systems. I would also like to acknowledge that his lectures on multi-agent systems were part of my inspiration to do the work in this thesis.

I sincerely thank the Republic of Singapore Navy (RSN) for providing the sponsorship that had made this academic opportunity possible.

Lastly, I would also like to thank Mr. Richard Black-Howell for putting in his time and effort to edit this thesis. Without him, I would have to spend much more time and hard work in getting this thesis completed.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

There are two main domains in modern simulation systems. The first is the Event Driven domain that uses events as the driver in the simulation. An event driven simulation progresses as events take place; usually, time is a measured result from the simulation system. This type of simulation is suitable when specific events can be pre-defined and mapped in the simulated world. Event driven simulations, specifically Discrete Event Simulator (DES), arguably bear a closer resemblance to the actual world. The second type of simulation domain is the Time Driven domain. In time driven simulation, also commonly termed as Time Stepping simulation, processes take place in accordance to time. At each time step, the simulation halts for the processing cycle to be performed by each entity. These entities, for example, can be autonomous decision-making agents, user controlled decision inputs, or environmental conditions affecting the simulated world. Time driven domain, due to its time stepping characteristic, has been favored by Agent Based Simulation (ABS) and likewise for Multi Agent System (MAS). This is due to the agent decision cycle, which can be easily correlated with each time step. Because human behavior is unpredictable, coupled with the randomness in human differences, ABS and MAS provide the ability to model what mathematical formulae could not. The differences and individual benefits of both the Event Driven and Time Driven domains have led to independent development and many achievements over the last two decades [1] – [4].

This thesis developed a new designing concept for building a more realistic simulation model. Inspired by both the DES and MAS behavior models, a new design concept to incorporate MAS behaviors into DES is proposed. It is not difficult to realize that the real world consists of events that trigger behaviors and decisions. Despite that, it is impossible to model the whole world, which is made up of a huge number of events taking place concurrently. It is, however, feasible to carefully scope and scale simulated worlds to be modeled with finite events. The proposed design concept targets the need to harvest "intelligence" of multiple autonomous agents interacting with a DES, and is given the term Discrete Event Multi Agent Simulations (DEMAS). This thesis exploits

the popular modeling methodology, Event Graph, developed by Schruben [5], and couples with the Listener Event Graph Objects (LEGO) developed by Buss [6], [7] as the designing tool in the DEMAS modeling. Simkit[1] is then used in the development to demonstrate and verify a few of the DEMAS systems introduced in this thesis as verification of the design concept.

It is not a new concept to hybrid both DES and MAS. As early as 1995, SWARM [9], a set of Objective-C libraries developed by Santa Fe Institute, provided the mean to implement agent-based modeling driven by continuous processes and discrete events. More recent researches included Dubeil and Tsimhon [10] and Wagner [11], works that looked at integrating ABS into DES, and Agent-Object-Relationship Metamodel respectively. Chapter II of this thesis consists of background understanding of DES and MAS. It can be read separately, especially for readers and researchers who want to know the fundamentals of each. It is also important to note that the materials covered in both DES and MAS will not be substantial and it is recommended that readers who need more information may obtain greater details from the references provided.

Chapter III looks at the research works done by Dubeil and Tsimhon [10], Wagner[11] and Riley[12], who all have proposed their methodologies to integrate DES and MAS, and summarizes some of the important findings. This chapter further highlights the reasons and benefits, together with potential usage, of having simulation models integrating multiple agents into DES. This approach is meant to result in more realistic simulators when carefully designed.

Chapter IV will discuss the DEMAS design concept using Event Graph and LEGO. A new graph figure is introduced to represent multi-agents decision cycle. The chapter will continue on to conceptualize DEMAS design concept by introducing two DEMAS examples. The first is a simple queuing system originated with DES, while the second example is the El Farol problem, which is a well known MAS example.

[1] Simkit is a set of Java packages which support the building of discrete event models. It was developed by Professor Arnold H. Buss of the Naval Postgraduate School (NPS) [8].

Chapter V describes the implementation of the DEMAS in the System Engineering and Analysis (SEA) Project 11 Logistic Support Simulation. The SEA11 logistic simulation was completed in June 2007, by the students from both the SEA curriculum of the NPS and the Temasek Defence Systems Institute (TDSI) of Singapore, The project studied the logistic support for U.S. 2012 Riverine Operations; the simulation was built using Simkit and functioned as a pure DES model. The addition of MAS in the simulation allowed autonomous decision making processes to achieve local and global goals. The results and analysis from the new SEA11 DEMAS are used to compare with the original model.

The conclusion of this thesis re-emphasizes the benefit of merging MAS into DES as the new design concept called DEMAS. When carefully modeled, a realistic simulated environment coupled with human-like intelligence and behavior can be achievable using DEMAS design concept.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND STUDIES

This chapter provides the background understanding of Discrete Event Simulation (DES) and Multi Agent System (MAS). The individual sections can be read separately as they are catered toward readers and researchers who need the fundamentals. The materials covered will not be substantial and therefore it is recommended that readers needing more information on the various fields obtain better coverage from the references provided.

A. GENERAL BACKGROUND

Simulation is a type of modeling approach used to study a system. A system here represents a real-world process of interest, or facility of a kind that warrants a scientific study [13]. A system is also defined by Schmidt and Taylor as a set of entities, such as people and machines that communicate and work as a whole in achieving some logical end [14]. Other than simulation modeling, there are generally two other types of model, namely, the Analytic and Algorithmic models. Analytic models are best suited for systems represented by well defined sets of equations, while algorithmic models are preferred in systems with logical characteristics that can easily be translated into computer algorithms. Simulation, on the other hand, is suited for systems with randomness, where no simple equations or logic are derivable. Both DES and MAS are types of simulation techniques used in modeling a system.

B. DISCRETE EVENT SIMULATION (DES)

The term discrete as opposed to continuous indicates that there is a moment of “pause” or breakpoint. And at each breakpoint, the State of the system changes instantaneously. Law [13] defines the state of a system as a collection of variables necessary to describe a system at a particular time. A complete definition of DES is therefore a simulation concerned with the modeling of a system, with representation at each separated point in time of the instantaneous changes in its state variables. Each point in time is said to be an Event, which may change the state of the system.

The structure of a DES model consists of four elements as shown in Table 1. These elements form the basic model of a DES [15]. To illustrate the difference of these elements, Figure 1 shows a diagrammatic use of each element in representing a system.

Table 1. Structure of a DES model.

Elements of DES	**Definition**
Parameters	Variables input into the system but will not change at each discrete interval.
State Variables	Variables representing the state of the system. These may change at each discrete interval.
Events	Interesting occurrences of the system.
Scheduling Relationships	Events' inter scheduling connection within the system.

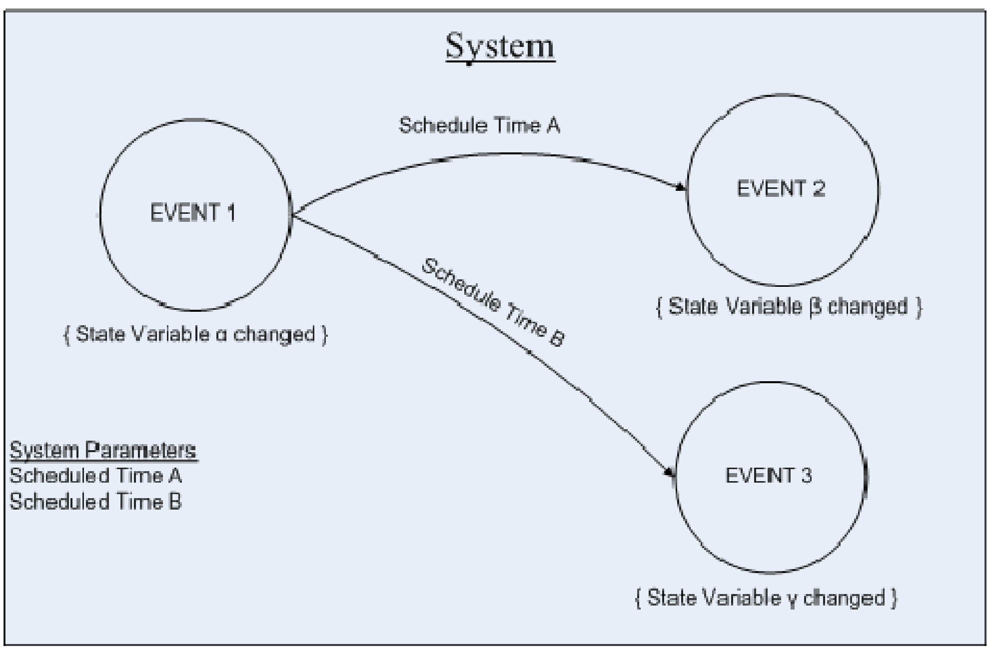


Figure 1. An Illustration of a Discrete Event System model.

Figure 1 shows a simple system with three events. Since the scheduled time A and B used in the system do not change the state of the system, they are termed the parameters. When each of the events takes place at some discrete timing, represented by

circles in the diagram, the state variables representing the status of the system get changed. α, β and γ are the state variables used in this system. The scheduling relationships shown in the diagram indicate that after Event 1 takes place, Event 1 will schedule the occurrence of Event 2 and Event 3 after scheduled time A and scheduled time B, respectively. To better understand the structure of a DES model, Figure 1 is mapped into a simple real-life Automated Wake-up System example as shown in Figure 2. Despite that, the given example is one that is unsophisticated, and therefore does not require the need of a scientific study; it is considered to be a complete DES model suitable for crafting into a computer simulation.

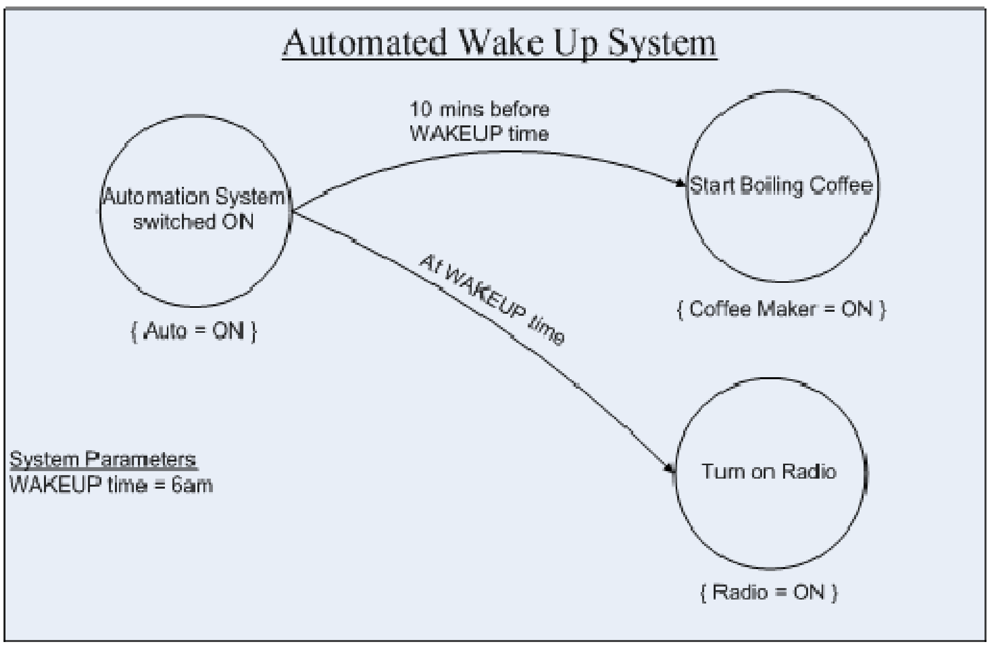


Figure 2. Simple Automated Wake-Up System.

As stated earlier, the automated wake up system example is a discrete event system with changes occurring at particular instants of time. This time is advanced at discrete steps to the next interesting state change, or a new event occurrence such as “Turn on Radio.” In general, it is not straightforward to see a real-life system as a discrete event system. The ability to do so takes practice to develop a good model.

C. MULTI AGENT SYSTEM (MAS)

To understand MAS, it is best to first grasp the concept of an Agent. An agent, in computer simulation terms, means a simulated autonomous entity - such as a person, animal, vehicle or the weather - that has attributes and potentially complex behaviors. The attributes and behaviors of the agent allow it to strive toward individual goals. Figure 3 shows a graphical representation of an agent illustrated by Professor John Hiles during his lecture in NPS. When many of these agents are placed within a system, it becomes MAS. The agents interact within the system and, depending on the setup, sometimes interacting among each other, with the sole objective of fulfilling individual goals. The scientific benefits achievable from MAS are vast [16]. Due to the ability to model complex behaviors in social context, MAS has recently gained interest among the defense and homeland security departments, in attempts to model and study terrorist behavior. Some examples of terrorist MAS research work are Bulleit and Drewek [17], and the NetBreaker [18] from North et al.

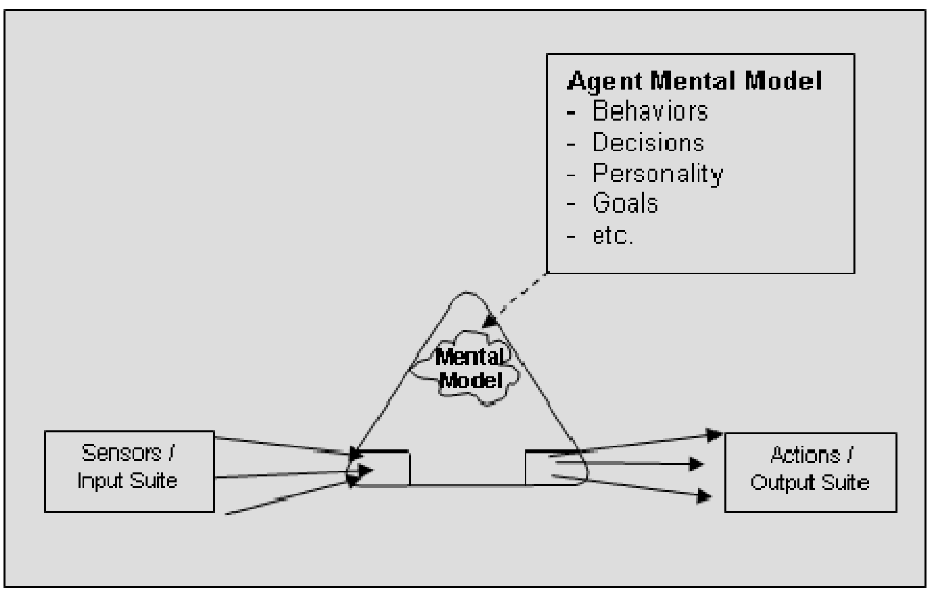


Figure 3. Graphical representation of an Agent by Prof. J. Hiles of NPS.

A MAS simulation typically consists of multiple autonomous agents with complex behavior, attempting to function in the modeled system. Agents will likely be

given individual goals that they will strive for in order to "survive" in the system. Each autonomous agent follows a time-stepping driver in making decisions for next action. At each time step, agents make decisions based on the information gathered from the sensors probed into the environment. The decision's outcome in turn becomes influential toward the system. This is the micro level observation of a MAS system. In the macro level, as each of the autonomous agents reacts to the system based on its individual goals, global objectives and behavior can be observed for study and analysis. MAS simulations consisting of agents with human-like complex behaviors will have the potential to exhibit global human community behavior at the macro level.

Professor Hiles describes MAS as a complex adaptive system [19]. To fully satisfy a MAS requirement, MAS needs to have an Environment, a group of Agents, a set of Objects, a set of Operations and finally some Laws. These are shorthanded as MAS = {E, A, O, Ops, Laws} [19]. Table 2. lists out the components of a MAS and the respective definitions.

Table 2. MAS Components and definitions.

Component	**Definition**
Environment	The environment is where the agents reside. It is a boundary to what is inside the system and what is to be left out. For example, does the system demand an indoor environment or an outdoor environment? For example, in a study to evaluate the elevator usage of a multi-story shopping mall, the environment is the building inclusive of elevators, escalators and stairs.
Agents	As explained, agents are autonomous entities, actively participating in the system. They have behavior and make decisions to achieve individual goals. Agents' action from the decisions made can include movement, sensing or modifying the environment. Agents can differ from each other in behaviors, goals, utilization of feedback, way of

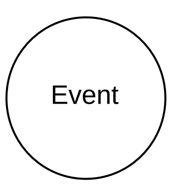
	seeing the environment, choices of alternative behaviors and even acting on the environment. In the elevator usage system, agents are customers who decide to use the elevators or to use the escalators. Decisions will base on the fact that escalator serve single floors while elevators serve more than one level.
Objects	Objects are the non-autonomous parts of the Environment. Objects can range from resources, tools, devices, concepts to even categories. Objects are utilized or acted upon by the agents. In many cases, unnecessary objects designed into the system may complicate the environment, which directly increases the complexity of the agents' interaction. In the same example, the elevators and escalators are the objects with which the agents interact to achieve goals.
Operations	Operations are the processes that operate within the environment. In most cases, operations are part of the agents in the way they interact with the environment. Sometimes operations can be more complicated and may reside within multiple agents. In the example, the utilization rate of the elevators will be gathered. Other operations such as average waiting time for the elevators and escalators can also be obtained through implicit agents' behaviors and decisions.
Laws	These are the ground rules that govern the environment of the MAS. These laws are the constraints that the agents must obey. The number of customers an elevator can carry, the service time of the elevator at each floor, the tolerance level of the customer for waiting on elevators are some examples of laws that the shopping mall MAS will have.

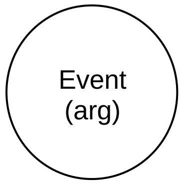
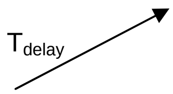
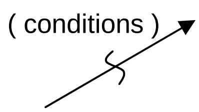
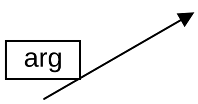
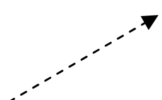
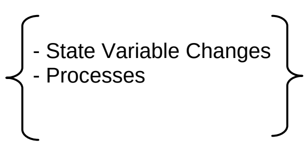
D. THE EVENT GRAPH METHODOLOGY

Event Graph was first introduced by Schruben as a concise method for organizing the elements of a DES [5]. The method was subsequently refined by Sargent [20] and Som and Sargent [21]. The simplicity and clarity of event graph methodology resulted in its popular acceptance in designing DES models. Figures 1 and 2 are actually event graphs that have expressed the concept of DES effortlessly. As reusability, modular designs and object oriented (OO) programming became popular, a greater breakthrough in DES modeling was achieved by Professor Buss, who introduced the idea of Listener Event Graph Objects (LEGO) in event graph [6], [7] and subsequently the development in the set of the DES Java package called Simkit [8]. LEGO, which allows modularity and reusability within DES, is the main driver that has allowed the design concept DEMAS in this thesis to work.

Event graph has direct representation for three out of the four elements of a discrete event system, namely, the state variables, the events that change the values of these state variables, and the scheduling relationships between the events. Parameter, which is the fourth element, exists within the discrete event system but is represented indirectly as unchanged values parsed within the system. Table 3 lists the basic symbols used in event graphs.

Table 3. Basic Symbols used in Event Graph.

Symbol	**Definition**
	The basic unit of an Event. This represents an event within the discrete event system at an instant of time, which state variables may change.

	An Event with defined argument parsed in by other scheduling events. There is no limit to the number of parameters parsed into the event.
	The Edge symbol that connects two events. This symbol means that there exists a scheduling relationship between the connected events and that Event A (arrow tail) will schedule Event B (arrow head) after time interval of T_{delay}. If T_{delay} is absent, it represents a schedule with $T_{delay} = 0$.
	The condition edge. Similar to the normal scheduling edge but the connected event (arrow head) will only be scheduled if the condition is valid.
	Scheduling edge with arguments parsing. This must correlate with the same arguments as the scheduled event.
	Cancellation Edge symbol used to remove a scheduled event if it existed in the future events list. Matching event name and argument if parsed ensures the correct event will be cancelled.
	The curly brackets, used under each event institute the state variable changes and processes associated at the discrete event time.

A simple Library Book Loan System will be able to demonstrate the use of the event graph methodology in modeling a discrete event system. To simplify the problem, the model will only deal with the number of books loaned out at any specific time. Figure 4 shows the event graph model of the Library Book Loan System, where the "RUN"

event acts as an initialization, which Schruben advocated. The “RUN” event schedules, without time delay, the “Borrow Book” event where library members at the queue will be assigned with the book they borrowed and the “Book on Loan” state variables get incremented. The “Borrow Book” event will schedule the “Return Book (Member)” event only after a time delay of T_{return}, which is a parameter (usually random) associated with the members. However, in order that the “Return Book (Member)” event knows which member is returning, the argument “Member” is parsed into the event. To continue serving the remaining members waiting in the queue, the “Borrow Book” event schedules itself with no time delay, with the condition that there are still members waiting in the queue.

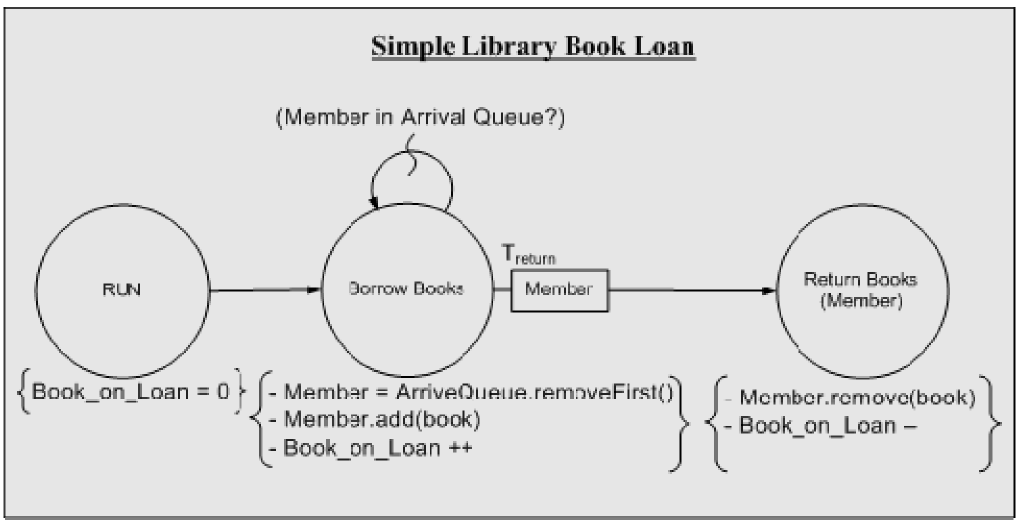
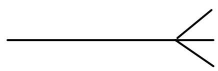
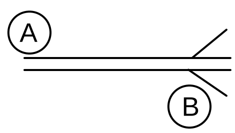


Figure 4. Simple Library Book Loan Event Graph.

LEGO is a powerful and flexible component framework developed by Professor Buss. LEGO promotes reusability and modularity within DES modeling [6], [7], which goes hand-in-hand with the present trend of OO programming. In the traditional event graph models, all events are pre-defined and hard-wired to represent a system. To expand the system, additional events cannot be added without having additional edges modified into the existing system. Such models eliminate the potential for reusability and encapsulation. LEGO bridges the gap by providing a listening link between separate

event graph models. A scheduled event in one model becomes the trigger to another event in other models. These listening links are listed in Table 4.

Table 4. LEGO Symbols used with Event Graph.

Symbol	Definition
	The Listener symbol, crafted to resemble the stethoscope used by a doctor. The symbol represents an Event Graph model listening (single line end) to the event triggers of the other Event Graph model (three lines end). Both events residing in both models must have the same names and arguments.
	An Adaptor symbol is a more complex LEGO framework. While a listener symbol is only concerned with event triggers with the same names and arguments, the adaptor symbol represents a listening link between two models that have the corresponding events labeled as A and B. In this case, the listening model listens to event B in the other model, and upon hearing event B being scheduled, event A in the listening model is scheduled.

The Simple Library example in Figure 4 is not complete without understanding how the "ArriveQueue" gets populated with members. Figure 5 shows the model of a Simple Library Arrival Queue event graph that models the member arrival at T_{member_arrive} (random valued parameter). "RUN" event cleared the queue and scheduled the "Member Arrive" event after delay time of T_{member_arrive}. "Member Arrive" event continues to schedule itself for the next member arrival and also the "Borrow Book" event, which in this case does not have a state variable changes.

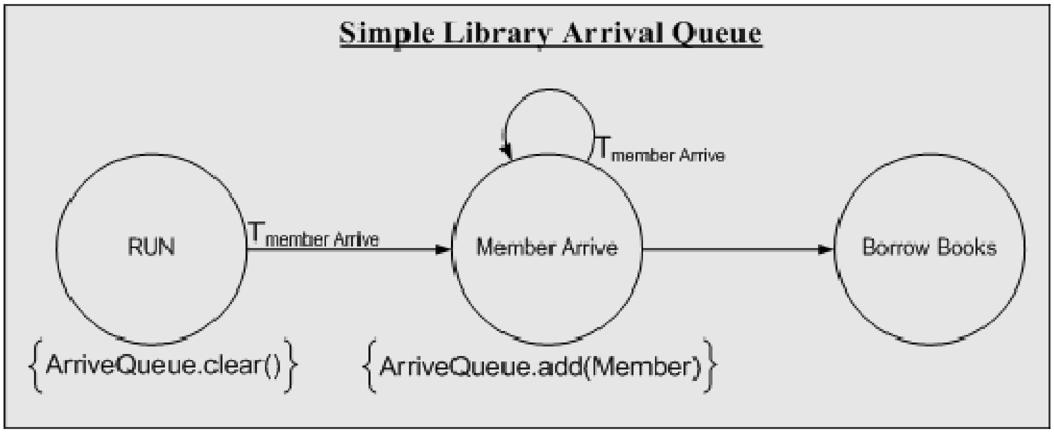


Figure 5. Simple Library Book Loan Event Graph.

Some people may ask why the Simple Library Book Loan and Simple Library Arrival Queue are built as two separate models. By separating the two models, it allows simplicity, encapsulation, modularity and reusability. Imagine that changes in one model no longer affect the other part of the bigger Library system. LEGO framework can now be applied to the two models by first using a Listener link as shown in Figure 6.

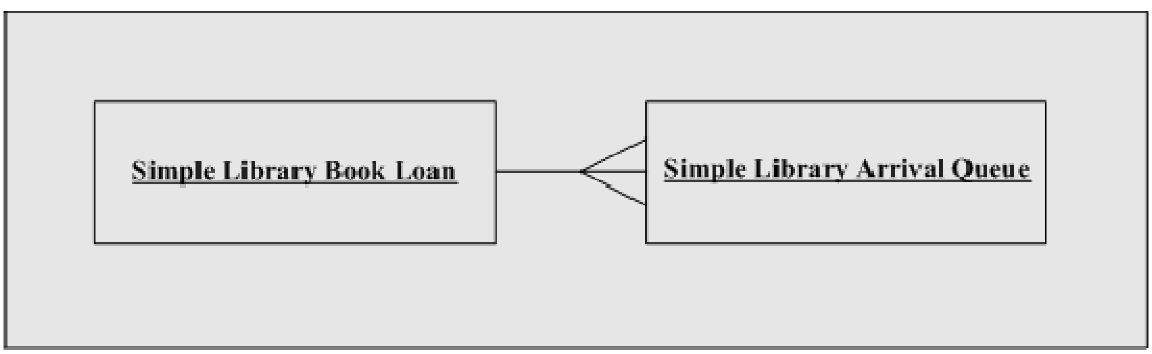


Figure 6. Linking Simple Library models with LEGO Listener.

To use LEGO Adaptor (see Figure 8) instead of the Listener framework, the "Borrow Books" event of the Simple Library Arrival Queue will not be needed, therefore simplifying the model further as shown in Figure 7.

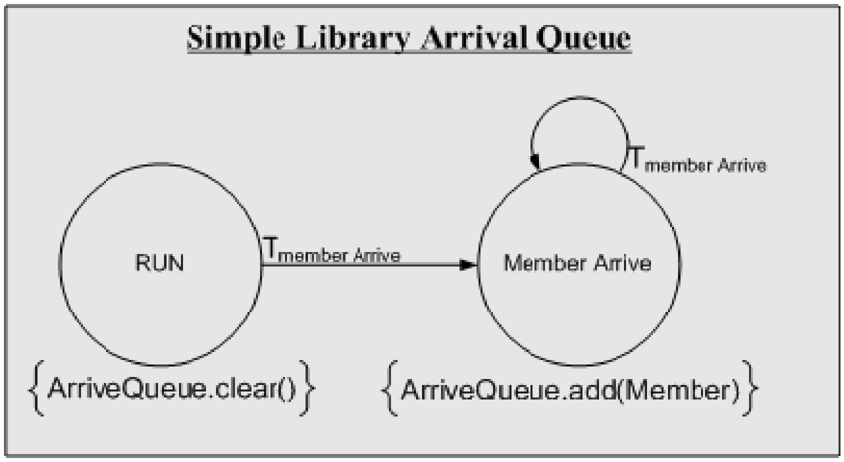


Figure 7. Modified Simple Library Arrival Queue Event Graph.

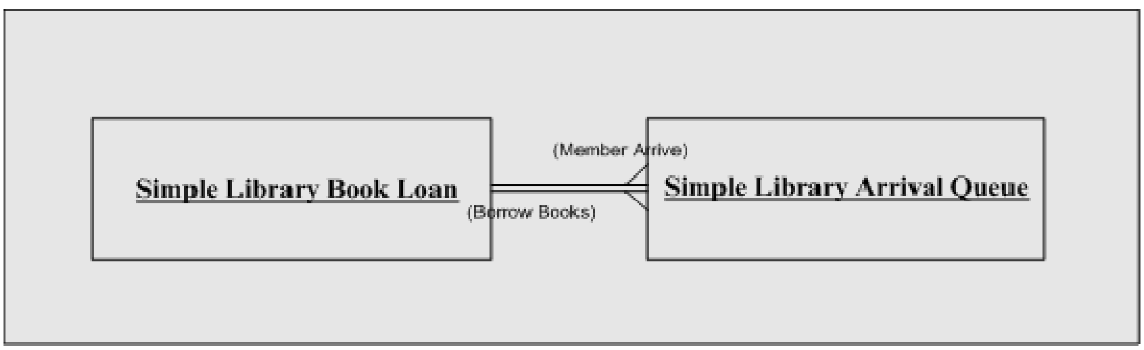


Figure 8. Linking Simple Library models with LEGO Adaptor.

E. SIMKIT

Simkit is an open source Java package created by Professor Buss. It is freely available for anyone who has the interest to build DES. Simkit was first introduced in 2001 [8]. This section is not meant to deliver a complete understanding to using Simkit, but rather a basic introduction. To be familiar with Simkit, it is recommended to attend the courses provided by Professor Buss in NPS.

Simkit has been designed to allow straightforward association with an event graph model. Every element in an event graph, including the LEGO framework (Table 3 and

Table 4) has a corresponding element in Simkit. Table 5 below shows the relationship between the basic elements of an event graph and Simkit classes, which was extracted from Professor Buss's class notes for OA3302, Winter 2007.

Table 5. Relationship of Event Graph and Simkit Classes (After Ref. [22]).

Event Graph	**Simkit**
Event Graph Parameter	Private instance variable, setter and getter
State Variable	Protected instance variable, getter, no setter
Event	`'do'` method
Scheduling Edge	Call to `waitDelay()` in scheduling event's `'do'` method
Run Event	`Reset()` method to initialize state variables; `doRun()` method to fire PropertyChange events for time-varying state variables.
Event scheduled from Run event	Call to `waitDelay()` in `doRun()` method
Event scheduled from any Event	Call to `waitDelay()` in scheduling event's 'do' method
Priority on Scheduling Edge	`Priority` instance as third argument to `waitDelay()`
Argument(s) on Events	Arguments in corresponding 'do' method
Parameters(s) on Edges	Add parameter values/expressions last (in correct order) in `waitDelay()`
Canceling Edge	Call to `interrupt()`
LEGO	**Simkit**

Listener Link	Linking model classes with `addSimEventListener()`. `A.addSimEventListener(B)` means that model B is listening to model A
Adaptor Link	First declare a new `Adaptor(“a”, “b”)` class and using then use `Adaptor.connect(A, B)` for the linking of model classes. “a” and “b” are the events in the classes while A and B are the model classes.

To implement Simkit components, new Java classes must abstract `SimEntityBase` class, available in the Simkit package. Figure 9 below shows an Arrival Process event graph. The corresponding extracted Simkit code is shown in Table 6.

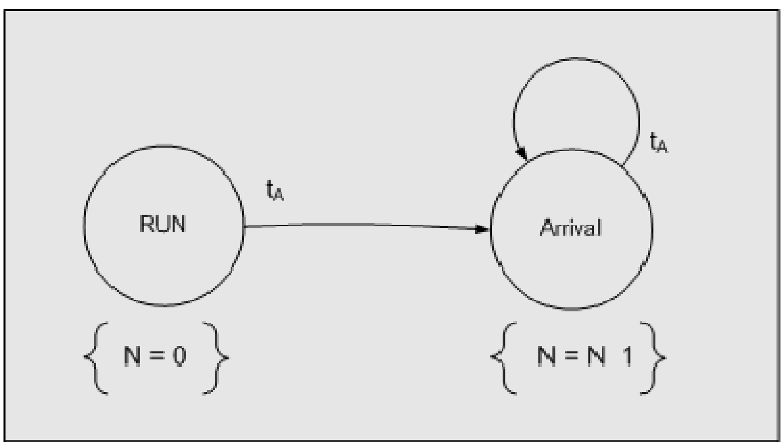


Figure 9. Event Graph for Arrival Process Model.

Table 6. Extracted Simkit Code for Arrival Process Model.

```
* Schedule ARRIVAL event after delay of inter-arrival time. */
public void doRun(){
  /** Initiate a print statement of the value change. */
  firePropertyChange("numberArrivals", getNumberArrivals());

  /** Cause a waiting time generated by random value. */
  waitDelay("Arrival", interarrivalTime.generate());
}

/** Arrival event method of the Event Graph Diagram.
* Increment number of arrivals, schedule next arrival after
* delay of inter-arrival timPeag.e 1*/
public void doArrival(){
  int oldValue = getNumberArrivals();
  numberArrivals = numberArrivals + 1;

  /** Initiate a print statement of the value change.*/
  firePropertyChange("numberArrivals", oldValue,
  GetNumberArrivals());

  /** Cause a waiting time generated by random value. */
  waitDelay("Arrival", interarrivalTime.generate());
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

III. LITERATURE FINDINGS

This chapter focuses on the literature findings obtained from the study of this thesis. The idea to hybrid DES and MAS emerged in 1995 from the Santa Fe Institute research group who developed the Swarm Objective-C libraries [9]. However, the merger concept did not gain further research popularity beyond the isolated effort to develop Swarm. It was not until 2003, that more studies and research papers were published. The following sub-sections will give a quick look at Swarm and its development up to now, followed by another three researches completed between 2003 and 2006. These findings allowed the addition to the developed work by other research and, more importantly, in support of the expected benefits achievable in this thesis research.

A. RESEARCH WORKS ON INTEGRATING DES AND MAS

1. Swarm (1995)

Swarm was built as a multi-agent software platform for the simulation of complex adaptive systems [9]. Swarm is not merely a collection of simulation modeling tools, but also provides a set of OO support libraries for analyzing the models and display functionalities, and also in controlling experiments on those models. A Swarm model will consist of a collection of independent agents interacting via DES, and each Swarm model can be hierarchically stacked into a nested structure, composed of swarms of other agents. The Swarm system is designed as a multi-agent system, where the basic unit is the agent. Each agent is any actor or entity within a system that can generate events that affect itself and other agents. The events generation and interaction among the agents are then handled by an “event scheduler mechanism,” which advances time based on the scheduling of these events. Swarm is an open source system available to the community of researchers building computer simulations.

The goal of Swarm is to provide consistent experimental tools in the form of libraries of carefully designed, implemented and tested code, targeted for reusability, sharing and encapsulation. Swarm has yielded many developments since its public

release in 1997. Swarm was first developed using Objective-C code in Linux/Unix environment, but recently, interfaces for C++ and Java, and installations suitable for Microsoft Windows and Macintosh OS have been made available. Swarm development is presently managed under the Swarm Development Group [23]. An annual conference, the "SwarmFest," is one of the events organized by the development group to gather Swarm simulation developers around the world. More research and development using Swarms is available in the informative SwarmWiki webpage listed as reference [23].

2. Agent-Object-Relationship (AOR) Metamodel (2003)

Wagner commented that Swarm does not support any cognitive agent concept, such as a theoretical foundation in the form of a metamodel [11], therefore limiting the specification of a Swarm simulation model to only low-level imperative programming languages. Wagner developed the AOR modeling language (AORML), basing it on high-level declarative specification language in Unified Modeling Language (UML)-based visual syntax (and an underlying simulation metamodel) coupled with abstract simulator architecture and execution model. AORML is claimed to refine the classical DES paradigm by enriching it with the basic concepts of the AOR metamodel [24]. Wagner proposed an Agent-based Discrete Event Simulation (ABDES) approach that adopted a number of essential ontological distinctions from the AORML with additional introduction of exogenous events.[2]

AORML is based on an ontological distinction between active and passive entities, with entity defined as: an agent, an event, an action, a claim, a commitment or an ordinary object in AORML, which can only interact through communications, perception of environmental information, performance of an action, making of a commitment and even satisfaction of claims. Objects that are passive entities do not have such capabilities. AOR modeling includes concepts for human (natural) agents, artificial (robotic) agents, and institutional (social) agents that are composed of numerous other autonomous agents. AOR modeling can also be modeled as external view or internal view, with the former

adopting the perspective of an external observer who is looking at the agents and their interactions while the latter express as a first-person view of a particular agent to be modeled. Figure 10 shows the core state structure modeling elements of external AOR diagrams extracted from Wagner's paper [11].

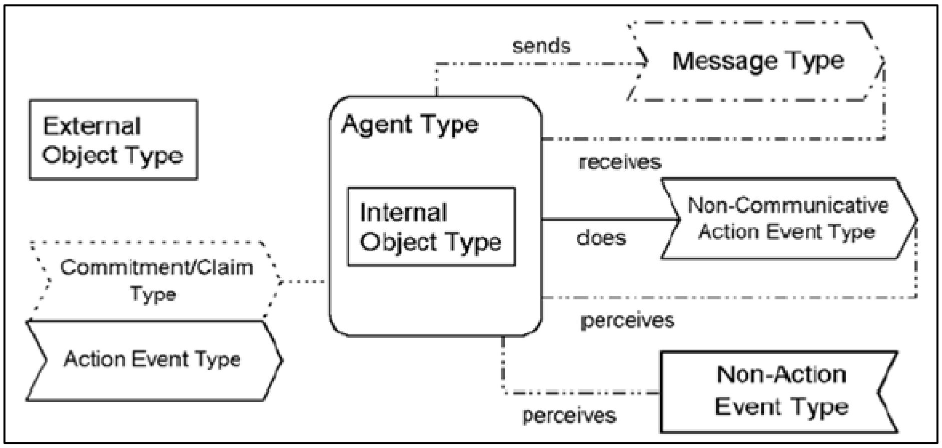


Figure 10. The core state structure modeling elements of external AOR diagrams [From [11]].

Wagner explained that in many ABDES approaches, similar to his, the basic DES model is modified to become an ABDES through representation of additional ABS conceptual distinctions, including the distinction between interacting agents and passive objects. The ABDES methodology - AOR simulation (AORS) - proposed by Wagner started with time-driven DES, but the listed simulation cycle steps, along with the set of PROLOG prototypes developed by Wagner, suggests that the simulation can be event-driven if desired. In AORS, the simulation modeling is achieved by AOR models and UML object diagrams, along with the encoding represented by means of high-level UML-based modeling language. DES controls the passive entities directly, influencing the active agents' internal state via event messages or signals and resulting in agents' responses via action event triggers that can be represented in an AOR metamodel. Figure

[2] Wagner explained exogenous events as non-action events that are not caused by other events or external action events in the sense that their actor is not included in the simulation, but generated periodically at random and by successor events, which are either caused event or action events [11].

11 shows Wagner's example of an AOR model with agent and event triggers, while Figure 12 shows the representation of a complete AORS as class diagram.

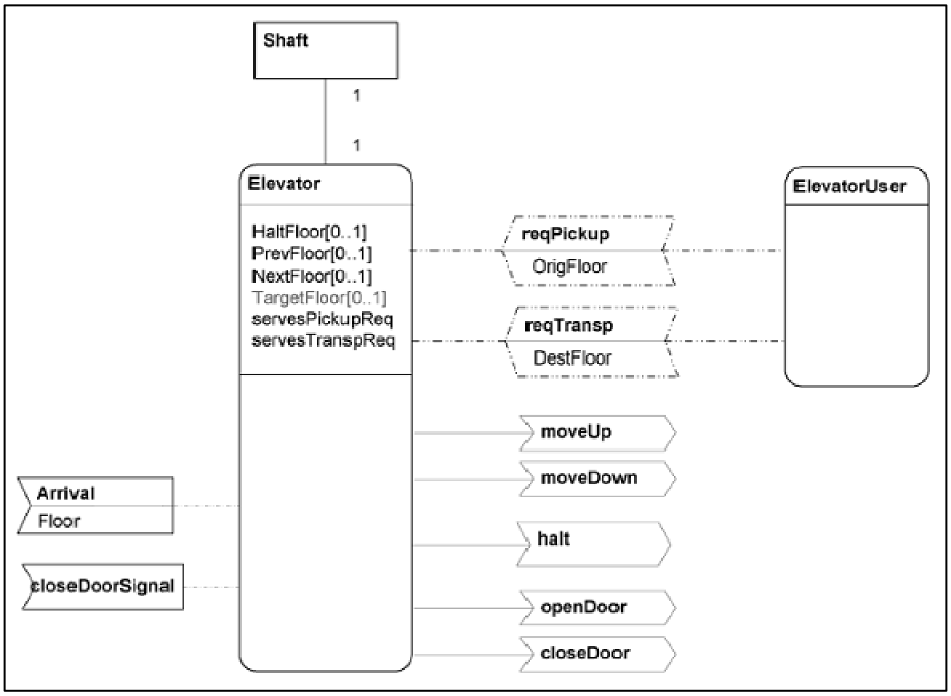


Figure 11. An Elevator scenario as an external AOR model [From [11]].

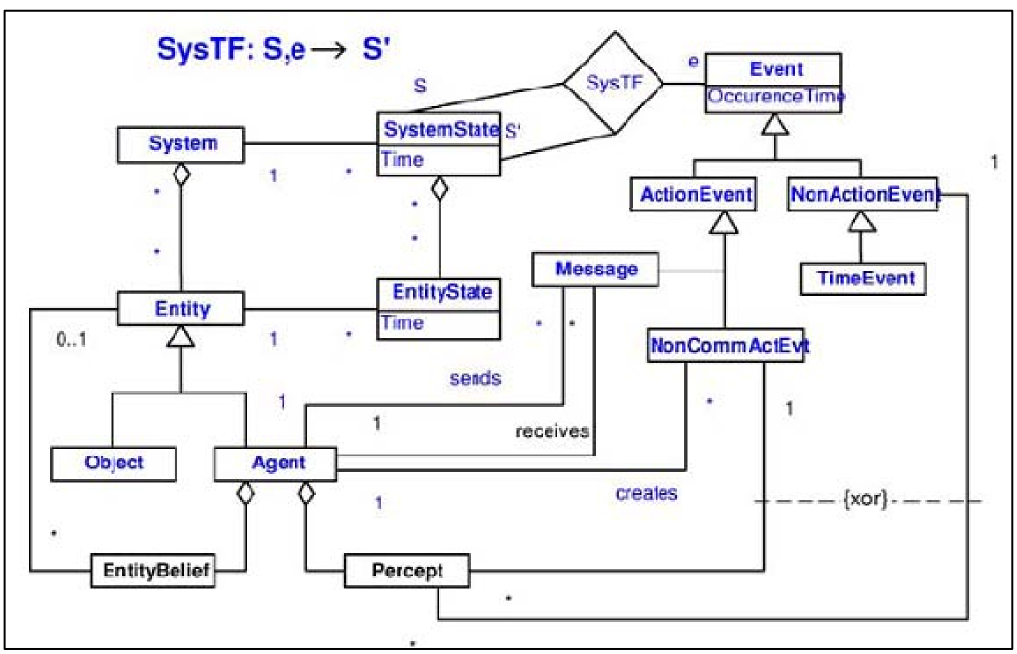


Figure 12. A UML class diagram describing the basic ontology of AORS [From [11]].

3. System for Parallel-Agent Discrete Event Simulation (SPADE) (2003)

SPADE functions as a middleware that primarily provides support for Artificial Intelligence (AI) simulations, with agents running in parallel across multiple machines. It also supports the tracking of computation time used by these agents. SPADE takes advantage of running the simulation backboned as a DES, therefore allowing a common association for SPADE to take care of the many system details required in handling the distribution in an efficient and reproducible manner. Developed by P. Riley [12], this middleware focuses on the agent as the fundamental simulation and is designed to support simulations for AI community without their being tied down to a particular simulated world.

There are five main capabilities of SPADE. First is the agent-based execution, with which SPADE is able to provide explicit support for modeling latencies in sensation, thinking and acting among the modeled agents. Second is the capability to distribute the agents across multiple machines, therefore increasing the pace of the simulation. The third capability involves result collection, where collected results are not affected by the network delays or load variations among the machines. In other words, the simulation results are independent of external factors. Fourth, the architecture of the agents is independent of the programming language used, allowing flexibility for developers. Finally, due to the DES engine's running as the backbone, the agent's action does not need to be synchronized in the domain. This eliminates any latency that may be caused by agents in different machines needing to wait for one another for actions to be executed simultaneously. Experimental results from SPADE indicate higher efficiency and significant parallel speedup in the agent simulation. SPADE is releases as a GNU Lesser General Public License with complete documentation downloadable at [25].

SPADE is very different from other researches to integrate DES and MAS. SPADE takes an agent-based simulation and layers it onto a DES backbone for easier distribution and control on multiple machines. In this way, DES in SPADE needs to be

transparent to the actual agent simulation. On the other hand, DES used in other integration researches focuses on harmonizing discrete events with agents in a complete simulation solution to the problem domain.

4. Integrating ABS into DES (2005)

Dubiel and Tsimboni highlighted that DES typically requires predefined paths with decision points that dictate entity movement [10]. Such limitation results in difficulty achieving human-like characteristic in path-finding. It was argued that any model that requires free movement of entities or a very detailed movement pattern is not easily simulated with DES. Two other limitations of DES involving human-like traveling characteristic include the impracticality of making decisions in very small time increments and the inability for entities to function autonomously. ABS, on the other hand, is a considerably better and straightforward approach to the problem domain, due to the possibility of simulating real-time interactions of people and their environment.

Dubiel and Tsimboni proposed a different approach in ABS and DES integration. In their study, ABS is integrated with DES to model humans' traveling freely through a DES environment. It was highlighted that the study will yield a desirable tool, given the ability to model scenarios including free moving, pseudo-intelligent individuals while utilizing existing commercial DES models.

A test case, based on a real-world problem from the theme park industry, was developed to validate the integration methodology. The scenario consisted of a customer's having limited knowledge of the surrounding searches and walks to an information source in an attempt to determine the location of a goal object, and then either traveling on a discrete movement system (tram) or walking to the goal object. Figure 13 gives the pictorial representation of the integration for the example used in the study. The agents were modeled with both visual and audio perceptions, along with a decision table mapped to choices, simple search ability, movement ability and also interaction ability in the test case. Figure 14 presented the physical layout of the test case used in the study.

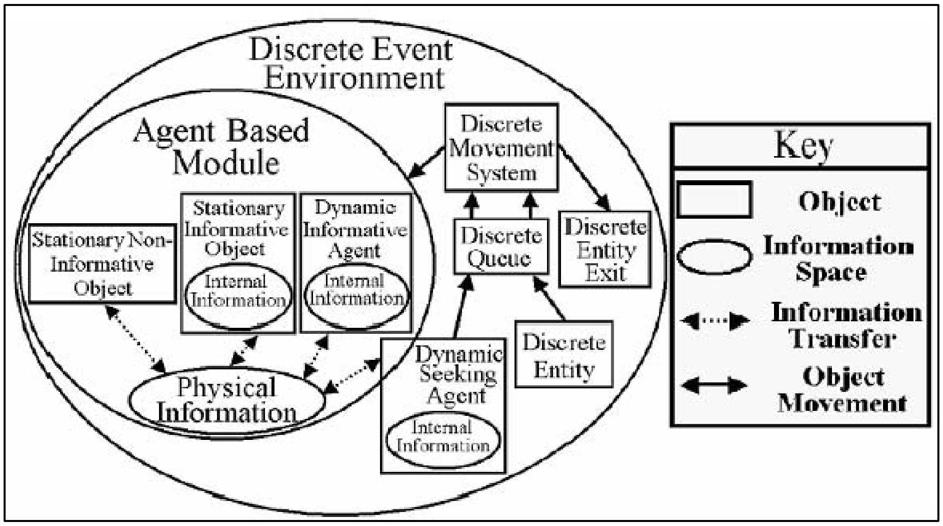


Figure 13. Integration of an Agent Based Module with the Discrete Event Environment [From [10]].

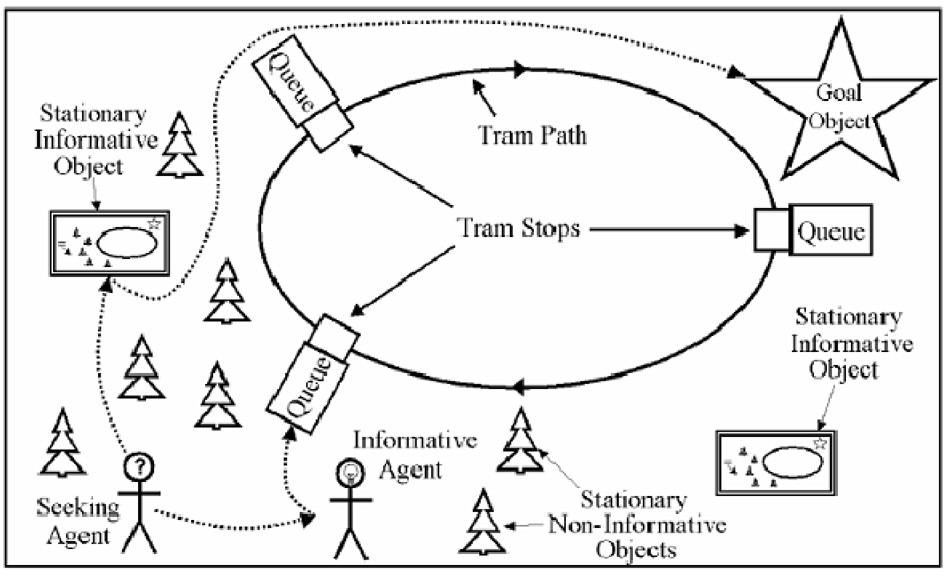


Figure 14. Physical Layout of the Environment and Some Possible Test Case Scenarios [From [10]].

Comprehensive verification and validation were done to study the results of the integration. The results indicated the successful implementation of the integration between ABS and DES, mimicking decision patterns and movement of individuals navigating the simulation world without a defined path or decision points. Nevertheless,

it was realized that the simulation domain, whether it is time stepped or event driven, was never mentioned. Through observation, it was implicit that the agents have to proceed in time-step decision cycles except when the agent enters the discrete movement system (tram). It was also mentioned that the agents, when seeking information, need to move in and out of the agent-based model to utilize DES functions. Arguably, Dubiel and Tsimboni's integration methodology provided a tool that allows agents to interact in traditional time-step cycle and the integrated discrete event processes.

B. BENEFIT STUDIES

After studying the various research papers and projects, it was clear that the benefits of integrating DES with MAS/ABS are worth pursuing. DES emerged as a simple simulation model that works well with most operation research (OR) and mathematical problem domains, such as queuing theories. On the other hand, MAS provides mean of modeling human-like complex adaptive behavior, a field that even today holds many unknowns due to the diverse possibilities involved. Table 7. Table 7 summarizes the benefits listed in the four research studies.

Table 7. Summary of Benefits cited in Research Studies.

Researches	**Benefits**
Swarm [9]	• Creates a set of efficient, reliable libraries for reusability. • Defines a structure for simulations, a framework from which models can be built. A DES of multiple agents using an OO representation. • Allows complex research simulations to be modeled with agents, but not tied down to any domain-specific requirement. Flexibility is achieved in using Swarm framework. • Complex agents modeling with time advancement based on scheduled events at successive times, eliminating unnecessary

	idling time-cycle in traditional time-step agent models.
AOR metamodel [11]	• AORS supports a structure-preserving modeling and closer-to-reality simulation that represents both passive entities (objects) and interactive entities (agents) • Functionally distributed simulation involving multiple participating simulators is possible due to the discrete event-driven engine. • Design flexibility that allows building of interactive simulation where the agent simulators may be replaced by real counterparts.
SPADE [12]	• Distributed simulation using multiple machines based on DES functionalities. Networking and correlations functions between machines are transparent to actual simulation. • Simulation architecture is independent of programming language. SPADE functions as middleware between the ABS and the background processing machines functioning as DES. • Large memories and processing time required by traditional ABS can be distributed to multiple machines, thus increasing efficiency and reducing latency.
Integrating ABS into DES [10]	• Provides an effective tool to implement human-like interaction and behavior into DES, especially existing commercial DES. • Integrating ABS into DES gives a more complete understanding of human-like decision pattern, behavior and interaction when modeling human path-finding. Integration allows merger in aspects of the system that cannot be simulated by either of the simulation methods alone.

The primary benefit in combining both DES and MAS/ABS is really to harvest the ability of modeling the different aspects capable by each simulation model. By doing so, simulations that better mimic a real world can be built for more accurate studies or as simulation trainers and gaming simulations. The advantage of using DES as the system environment model or as backbone of the simulation provides a simpler and more efficient architecture, both because events in DES can be well defined (and easily simulated) functions and because event-driven time advancement eliminates the idling time-step cycle that may happen in traditional time-step simulators. Carefully scoped simulated worlds consist of well defined events happening at discrete times, matching the idea of using DES to model and simulate the environment in these researches. With such a well defined simulation world using DES, layering MAS/ABS on top adds the intelligence aspect. In other words, expected applications from these hybrid models other than SPADE, which uses DES for distributed processing purposes, include systems with intelligence entities (MAS/ABS) interacting within a well defined environment (DES). Applications, for example, can be some commercial queuing and trading systems, military simulations for studies of terrorist behavior and mission planning simulators, and even simulators for biology and ecology research.

C. EXPECTED CHALLENGES

There are also challenges when merging both DES and MAS/ABS. The main challenge is choosing between an event-driven or a time-driven domain engine. This is the main difference, which has also given both simulation models their unique purposes. Table 8 lists some of the expected challenges when exploiting the work done in each of the four researches. These challenges will assist as guidelines to be considered when approaching the design concept proposed in this thesis.

Table 8. Expected Challenges of the Research Studies.

Researches	Challenges
Swarm [9]	• Java and C++ were not popular during the initial development in 1995; therefore, Objective-C was used for OO approach. Limited Java and C++ interfaces have been developed to use with Objective-C, but full compatibility is the challenge. • Lacks design tools. Only consists of standard libraries to model ABS.
AOR metamodel [11]	• Random, non-action events need to be represented; this is done in AORS by introducing the exogenous event. • The choice of event driven or time driven is left to developer using AOR. Even though the metamodel allows choice of either, the low-level design and implementation is not highlighted in this high-level AOR metamodel representation. • Knowledge of UML is needed before simulation modeling can take place. • Modularity and reusability are not the concern of AOR metamodel and AORS design.
SPADE [12]	• Functions as a middleware, which also means that the DES backbone architecture cannot be modified or customized.
Integrating ABS into DES [10]	• Elaborated path-finding design is unable to break away from time-step cycle required by MAS/ABS during path-finding. • Absence of design concept and guidelines to the integration since the paper is meant to only prove that ABS can be integrated with a common DES.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DEMAS DESIGN CONCEPT

This chapter introduces the Discrete Event Multi Agent Simulation (DEMAS) design concept. Similar to ABDES in the aspect that it is a hybrid of DES and MAS, this proposed design concept, however, focused on a different standpoint in the design. This standpoint will be introduced and concretized with the use of event graph methodology and LEGO concept, along with a new graph figure to represent the multi-agent decision cycle. To conceptualize the DEMAS design concept, two DEMAS examples will be shown. The first is a simple queuing system originating in the DES field, while the second is the El Farol problem, which is a well known MAS example.

A. DEMAS DESIGN OVERVIEW

The greatest different in the DEMAS design approach is really the main reason to use the acronym DEMAS as opposed to ABDES. Other ABDES, such as Swarm, AORS and SPADE, are agent focused. AORS, for example, focuses on external and internal views of the agents design. The DEMAS design approach aims at providing a balance between DES and MAS; however, with the starting base built on DES, this explains why "DE" is used as the first two letters in the acronym.

Studies from various researches highlighted in Chapter III indicate that a good design concept to hybrid both DES and MAS must, first, have flexibility, which means that the concept must not be confined to any DES or MAS structures. Second, the design concept must support modularity and reusability so to benefit future researches. Third, the design concept should not nullify any benefits given by both DES (simplicity and efficient events driver) and MAS (complex adaptive behavior modeling). On the other hand, the expected challenges from the various researches also emphasized that there should be robust and simple design tools catering for both DES and MAS as a system. The design concept in DEMAS uses event graph methodology because of its powerful, yet simple representation of DES [26]. Moreover, modular and encapsulation design are two crucial factors that event graph coupled with LEGO concept can achieve. Using event graph methodology will allow flexibility for DEMAS to be implemented in any

simulation programming languages. Nonetheless, implementing DEMAS using Simkit appears to be of great ease and advantage, given that Simkit has been solely developed to closely associate with event graph methodology and LEGO concept.

1. Model DEMAS by First Scoping the Simulated World With DES

Exploiting the modularity and encapsulation capabilities provided by LEGO [6], [7], designing the environment or the simulated world and the agent structure in DEMAS can be done in separate modules, either concurrently or otherwise. Nevertheless, it is highly encouraged that the DEMAS design approach should start with making the right discrete events world before linking the triggers that the agents perceive and interact with. To simulate the world with which agents interact, the "world" should first have well-scoped events taking place. In Professor Hile's concept where MAS = {E, A, O, Ops, Laws}, both Objects and Operations exist as part of the modeled Environment, and Laws are the binder that govern between the Environment and Agents [19]. The Environment design must therefore be self sufficient in its design with events manipulating the Objects and Operations, closely obeying the defined Laws. In DES perspective of this Environment design, the Objects, Operations and Laws are merely the state changes to the system via events, and these events will include both those that interact with the agents and those that do not. By defining the scope of the Environment first, trigger events that will interact with agents can be identified easily and mapped.

The best way to introduce the design approach to DEMAS is to go through an example. The following figures illustrate the approach of the design concept. To begin, Figure 15 provides the various events and an agent that the system will model.

Figure 15. Identifying the Events and Agent of a system.

In this simple system, there are three subsystems illustrating that a simulated world can be made up of encapsulated, modular subsystems in DEMAS. Each subsystem can be independent, or can rely on the events scheduled by other subsystems. In this example, the alarm clock subsystem will schedule a coffee timer event in which the coffeemaker subsystem will "listen" to, assuming that the coffeemaker is electronically linked for automation from the same alarm clock used to wake up the agent. The Bus subsystem is independent from the other two subsystems. After identifying the events within the system, the next step is to determine which triggering events the agent perceives and what kind of actions is expected, as shown in Figure 16.

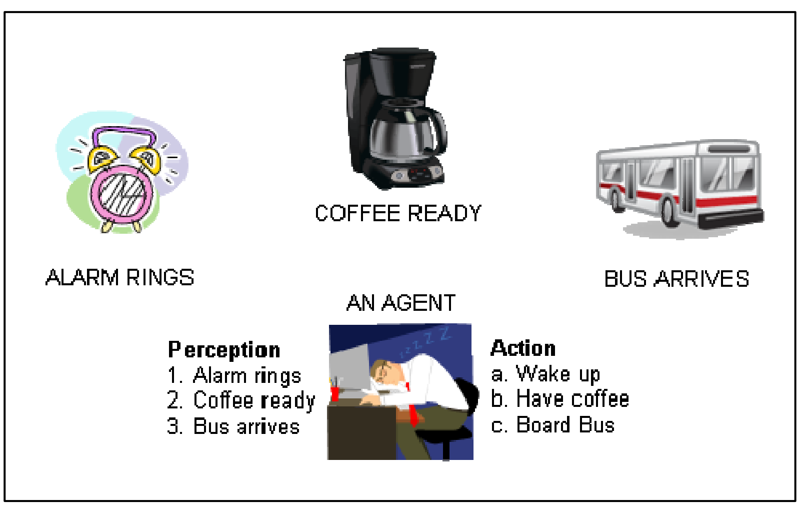


Figure 16. Identifying Perception and Action of the Agent interacting in the system.

2. Define the MAS Mental Model

The mapping of perception to actions, in this case, is the mental model of the agent. In some cases, the MAS mental model can be complicated. Agents can have single perception events that trigger off multiple action events interacting with the environment, or may involve sophisticated decision processes to derive which action events to take place. The use of the DEMAS design approach does not limit the mental model structure used. In general, as long as the intelligent system to be modeled involves an input suite and an output suite that interact with a DES world, the DEMAS design concept can be applied. For example, expert systems, neural network, fuzzy logic and even cognitive blending concept can all be used to model the intelligence of a DEMAS design, instead of the MAS mental model. However, this thesis focuses on using MAS as the intelligent part of the design, mainly because MAS is gaining popularity in both commercial and defense industry to model human complex behaviors.

Using a pseudo-event graph representation, the DES world of the DEMAS can be crafted as shown in Figure 17, with each subsystems clearly boxed out.

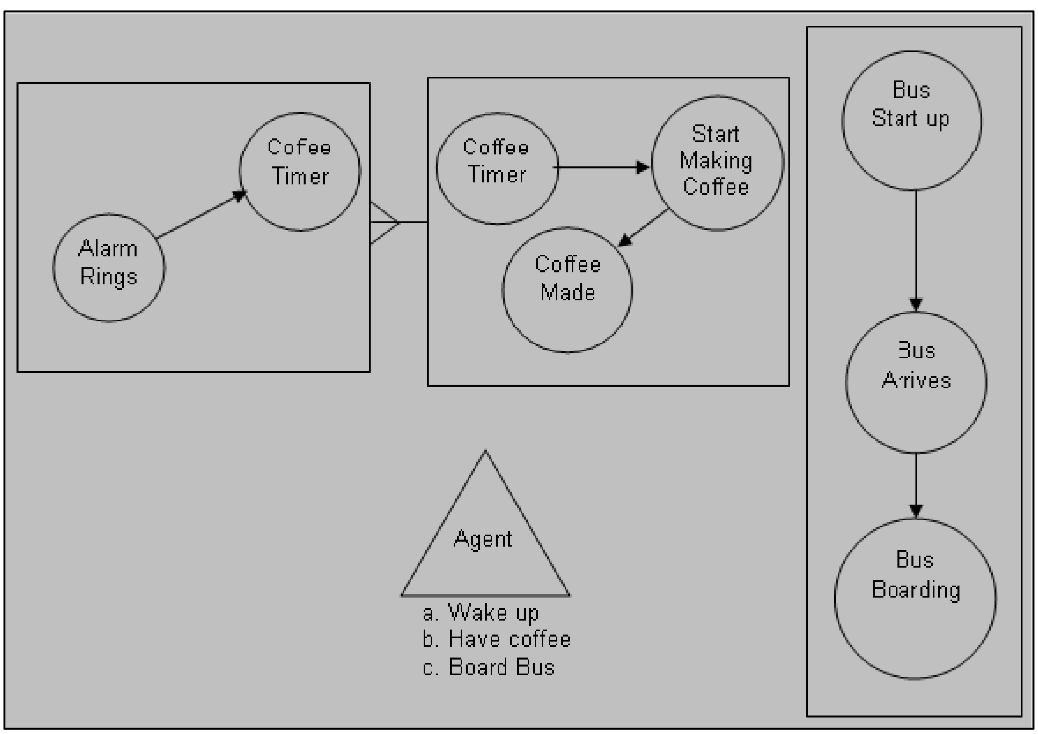


Figure 17. Pseudo Event Graph representation of the DEMAS model.

In DEMAS design, agents are represented by a triangle symbol in the event graph. The triangle shape is chosen because it represented the three important elements of an agent, which are the Input Suite, the Mental Model and the Output Suite, as shown in Figure 3.

3. Correlate Discrete Events in DES with MAS Mental Model

In the design the two actions, namely "Wake Up" and "Have Coffee," are not modeled in the DES environment because they are simply part of the agent states/conditions. Only "Board Bus" is an action event that interacts with the environment. With the DES environment crafted and the agent perceptions, mental model and actions identified, the perception events are then mapped as the agent's input suite and likewise the action event mapped back into the Environment as agent's output suite. These arrows represented scheduling of events, as shown in Figure 18.

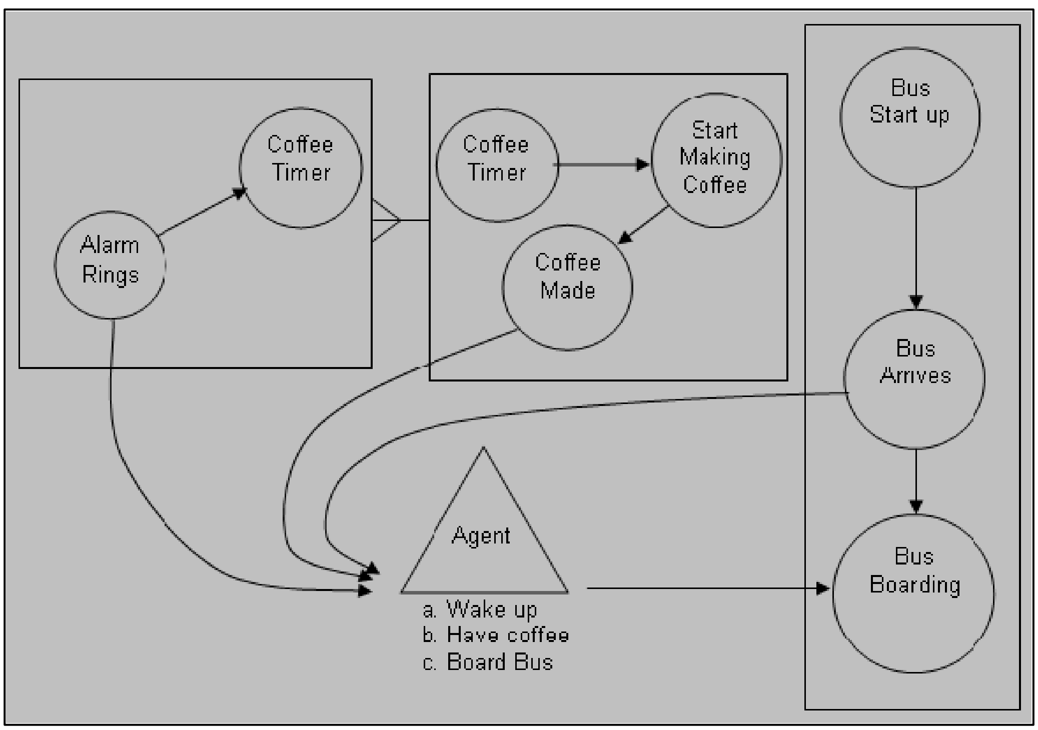


Figure 18. Events Scheduled as Input Suite and from Output Suite of the Agent.

The scheduling flow in Figure 18 is straightforward and self explanatory for a discrete event system. This is again the reasons for using event graph methodology in the DEMAS design concept. From the overview of this simple DEMAS modeling example, it is important to understand that the discrete event environment needs to be defined firsthand such that the simulated world will be event-driven focused. The time-driven mindset of the traditional MAS should be avoided at all cost, as it will defeat the benefit of having an efficient event-driven environment in DEMAS design.

B. DESIGNING DEMAS WITH EVENT GRAPH AND LEGO

The DEMAS design concept starts with an event graph that models the simulated environment in which the agents will “live” and with which it will interact. The simulated world should consist of smaller subsystems with encapsulation and modularity design in mind. The LEGO concept provided the Listener Link between these modular models such that scheduled events in one model will trigger off the same scheduled events in other listening models. The LEGO concept was explained in Chapter II of this thesis.

There is no difference between designing DEMAS with event graph, and designing normal DES with event graph. The only exception is with the introduction of the Agent Triangle symbol, made to resemble the agent configuration illustrated in Figure 3. Figure 19 presents an illustration of a complete DEMAS event graph model, consisting of three DES Subsystems and a MAS, using LEGO Listener Links to listen for event schedules in one another. The MAS system perceptions and actions triggers, likewise, use Listener Links for interaction with the DES subsystems. It is also possible to model more complicated DEMAS systems that consist of more than one MAS, as shown in Figure 20.

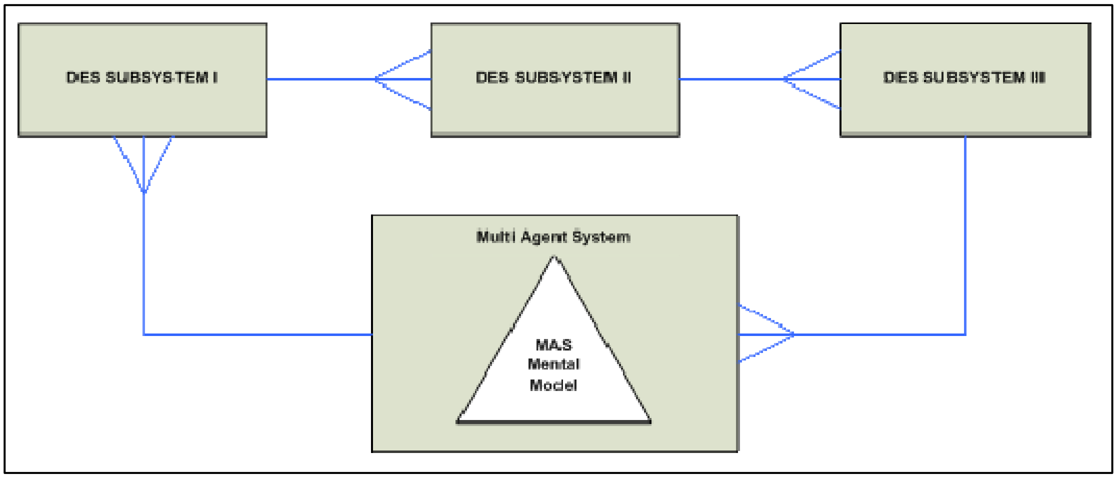


Figure 19. An Illustration of a Complete DEMAS Event Graph Model.

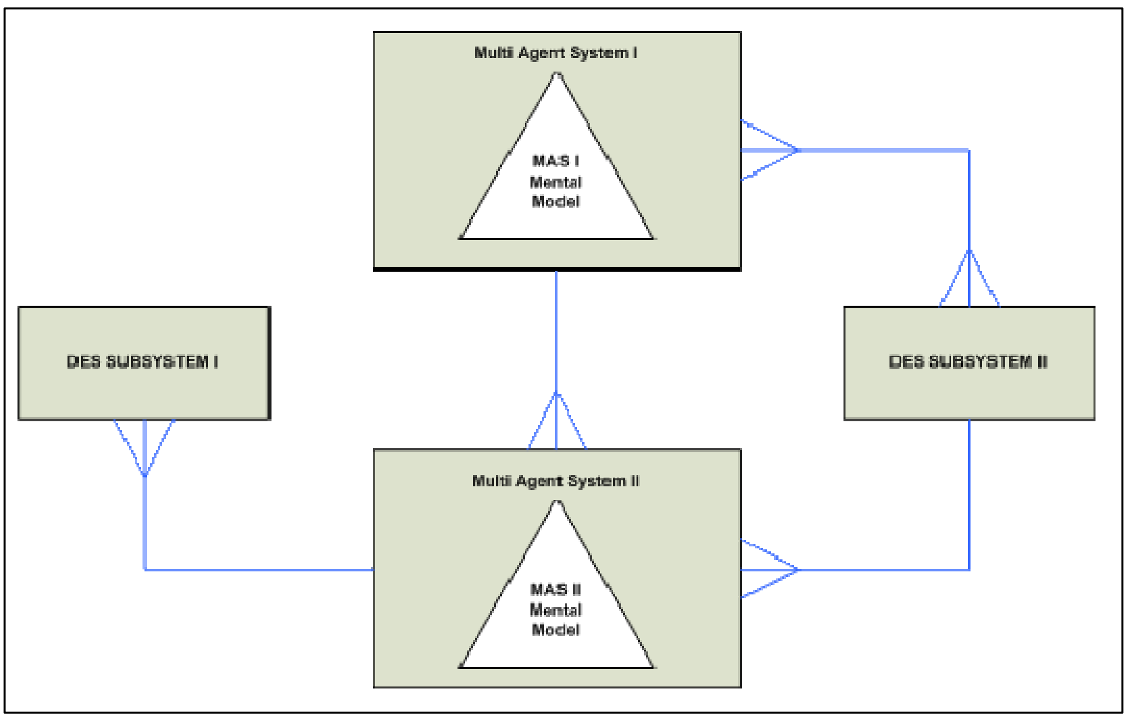


Figure 20. An Illustration of a DEMAS Event Graph Model with multiple MAS.

In the AOR Metamodel research introduced by Wagner [11], the example is given of an elevator scenario focused largely on the agent aspect with discrete events providing signals to trigger the decision process of the elevator agent (Figure 11). The simulated world supposedly modeled as discrete events was not elaborated in the research paper. To

verify the capability and flexibility of the DEMAS design concept, the elevator scenario from Wagner has been borrowed and re-modeled as the DEMAS design shown in Figure 21.

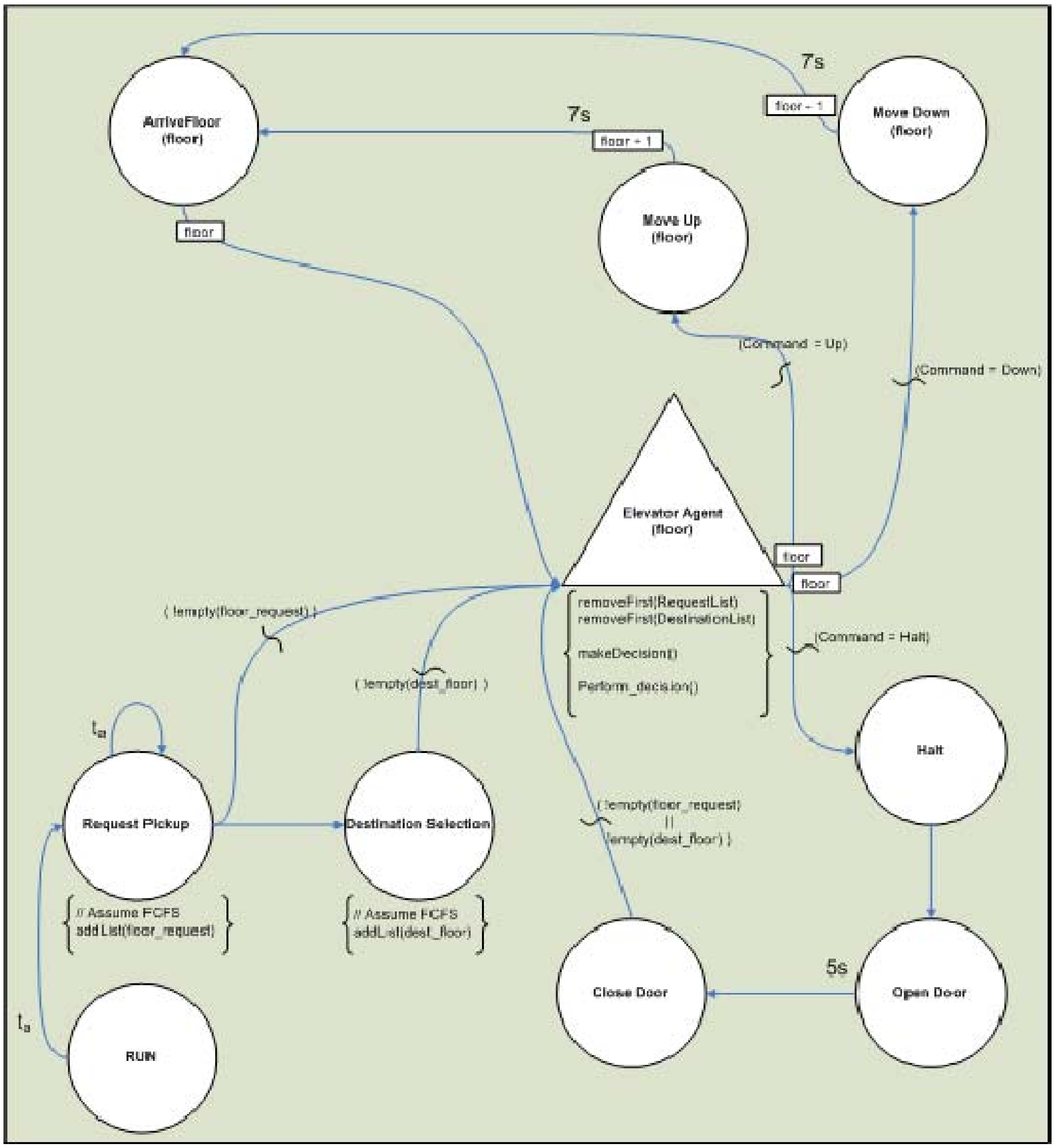


Figure 21. Modeling AOR Elevator Problem (Figure 11. Figure 11) with DEMAS.

Following the DEMAS design concept approach, the re-modeling starts with the elevator environment. From the study of the elevator AOR metamodel, information with regard to the signals triggering the elevator agent was analyzed and rebuilt as event

graphs. The elevator agent's decision processes given in the AOR design were meant to be at an abstract level, and likewise this applied to the re-modeled agent mental model in DEMAS. In the DEMAS design, the "Request Pickup" and "Destination Selection" events add floor services request to the list that the elevator agent process. The "makeDecision" process performed by the agent, not elaborated in detail, simply applies a decision-making cycle to ensure the most efficient floor stop at all times. The agent, upon determining the floor to service, will schedule the respective events, "Move Up" or "Move Down" until the desired floor is reached. "Halt," "Open Door" and "Close Door" are the various events that the agent will schedule upon reaching the desired floor. These scheduled events directly affect the operation of the elevator in discrete event-driven time advancement. One difference between the re-modeled DEMAS with the original AOR is the insertion of passenger arrival time "t_a". Adding the passenger arrival time increases the realistic modeling of the elevator scenario using DEMAS. The arrival time of the passenger can be assumed to be normally distributed around two peaks; the first is at the beginning of a working day and the second is at the end of the same working day.

C. DESIGNING THE MAS IN A DEMAS MODEL

On top of the agent triangle symbol added to the event graph methodology to represent MAS in DEMAS, a more complex MAS structure can be illustrated with the addition of a pictorial sketch. The use of the sketch is because MAS most often involves complex behavior that cannot be easily exposited with mathematical equation or logical expression. Using pictorial representation commonly termed as the "Helicopter View" or "Top View" was also advocated by Professor Hiles [19] in the MAS lectures conducted at NPS. The focus of the pictorial representation is to show the main piece of the agent mental model and how it relates with the scheduling events entering as input suite and exiting as output suite as a whole. Importantly, there is no specific guideline as to how the sketch should be drawn as long as the sketch is able to bring out the MAS structure.

To give an example of a MAS sketch, Figure 22 lays out the MAS structure of the well known El Farol Bar Problem. The El Farol Bar problem is a well known MAS example, which will be re-modeled with DEMAS later in this chapter. As a short

introduction, the El Farol Bar plays lovable music enjoyed by the 100 patrons. Nonetheless, the attendance threshold of the bar is defined as sixty patrons each night in order for anyone to enjoy the music. Setting the rule that there will not be prior communication among the patrons, each patron has to rely on their set of predictors to make the decision to go or to stay at home. The happiness of the patron is further governed by each of their personalities, which in turn gives weights to the predictors used. The objective of modeling the El Farol system is to determine if the patrons will eventually exhibit some complex adaptive behavior.

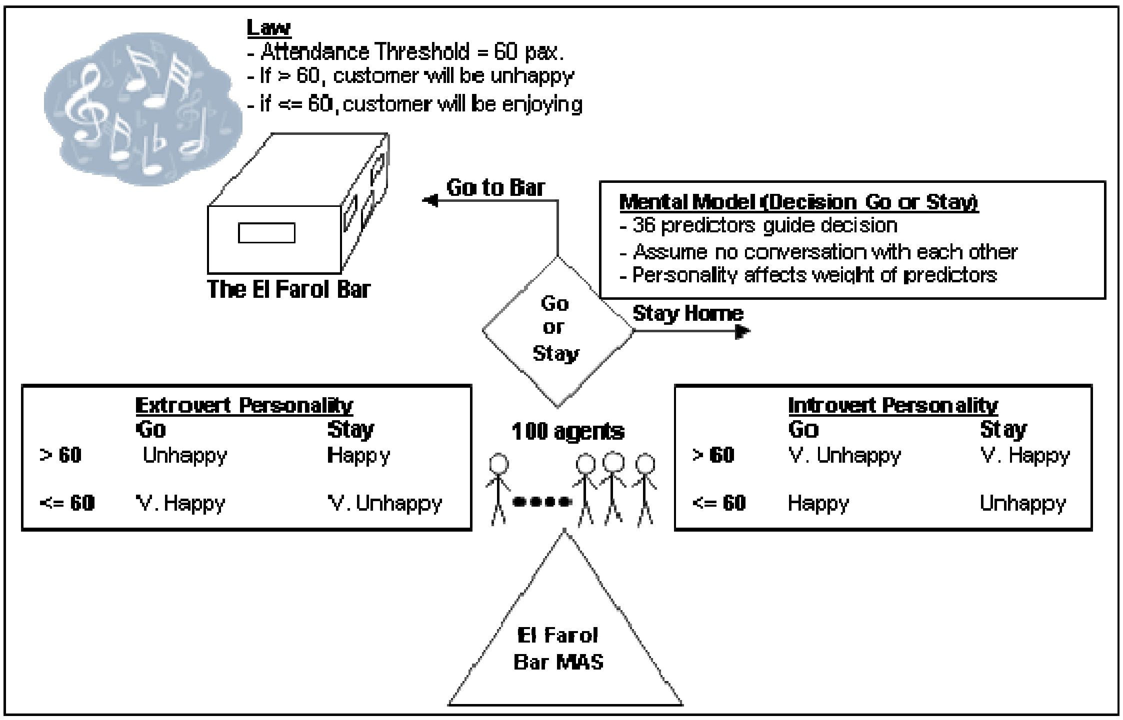


Figure 22. Pictorial Representation of MAS structure for El Farol Bar Problem.

D. DEMAS DEVELOPMENT USING SIMKIT

To verify the usability of the DEMAS design concept, two examples were tackled and re-modeled as DEMAS and implemented using Simkit. As mentioned, Simkit is built with great emphasis on implementation of DES designs using event graph methodology. The first example taken from Queuing Theory is the model of a Simple Server system involving customers arriving at the queue waiting to be served. The Simple Server system

will be incorporated with a MAS to determine if such added "intelligence" will improve the queuing system. The second example is the El Farol Bar problem, originated as a well-known MAS problem. The El Farol Bar problem will likewise be re-modeled into a DEMAS system and the advantages of doing so will be studied.

1. Simple Server DEMAS

Figure 23 is an event graph model of a Simple Server Queuing system. The Simple Server system is a standalone DES. As the system runs, customers arrive at t_a interval, which is usually dictated by a Normal distribution. The simple server system does not model rejection, thus all customers who had joined the queue will be served until the end of the simulation. When there is a free server available in the system, the first customer in the queue will be served and the service time again will be normally distributed, represented as t_s in the event graph. The time when the customer joins the queue to the time the same customer has been served is the total System Time obtainable from the simulation.

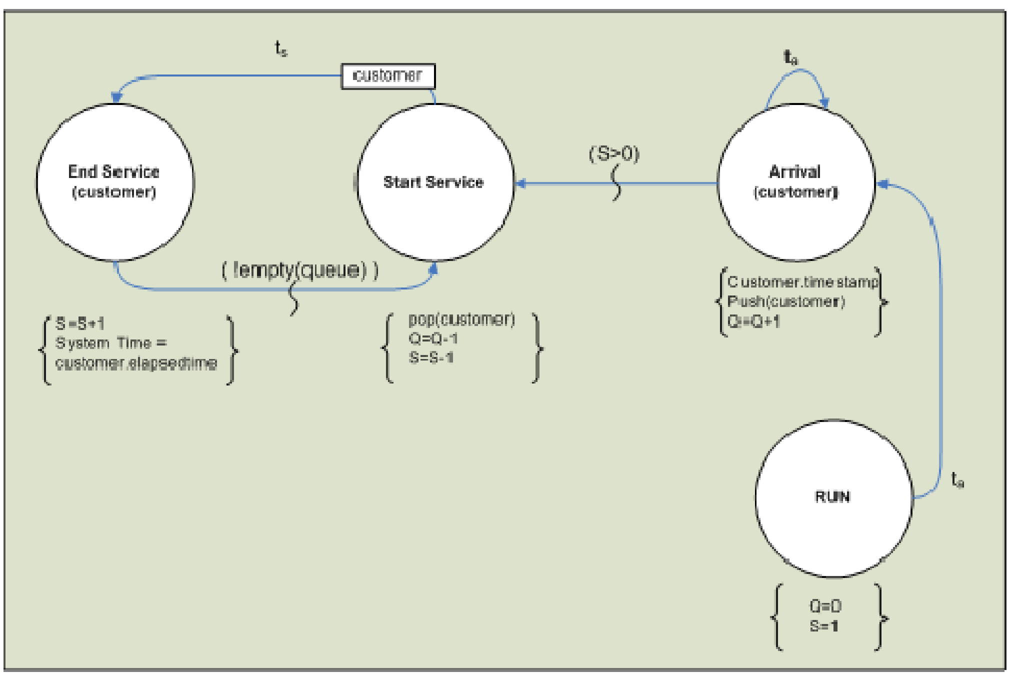


Figure 23. Simple Server Modeled with Event Graph.

To include intelligence into the Simple Server system, a simple MAS is added into the existing Simple Server DES shown in Figure 23. The main objective is to determine if certain adaptive behavior will be displayed if customers are allowed to vary their own arrival time individually. This is the desire to achieve the modeling of a more realistic queuing system, given that actual humans do vary their own preference to visit a service based on past experience. For instance, when a customer visits a bank on a Monday peak morning and realizes the waiting time is extremely long, he or she will vary his next visit to avoid the Monday morning peak. This is a demonstration of an adaptive behavior toward the system. Figure 24 shows the new DEMAS event graph added with MAS into the previous DES. The addition of the MAS into the DES does not cause much change, except for the re-routed edges into and out of the MAS. Modularity design with LEGO can be used but has not been applied because of the simple system configuration.

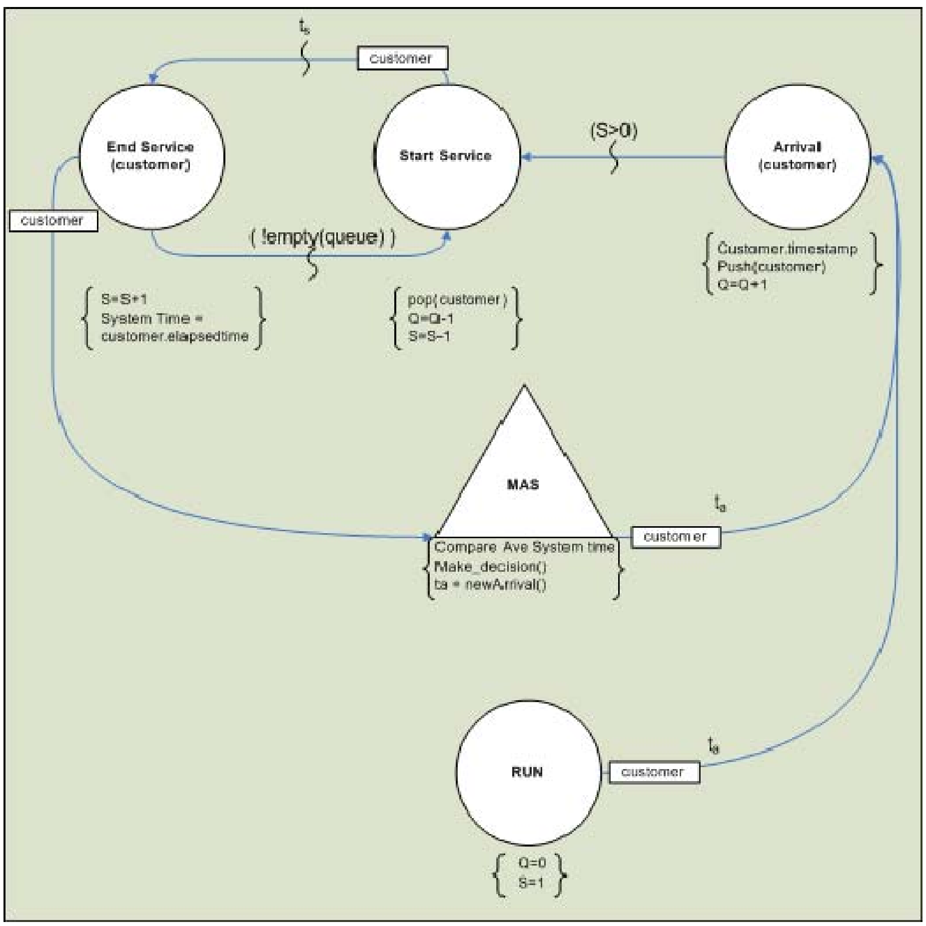


Figure 24. Simple Server DEMAS Modeled with Event Graph.

In this Simple Server DEMAS, the customer agents are first assigned a random arrival time with exponential distribution. It is important to note that customers within the server system advance in event-driven paradigm and, likewise, the customer decision cycle in MAS is also triggered via event scheduling. Upon completion of the service through the server system, the system time of each customer is remembered individually. Without interacting with other customers, each customer is then allowed to make a decision on the next arrival time, based on the consolidated average system time of all customers. The goal of the customer is to achieve a system time equal to or less than the average system time within the DEMAS system.

Implementing the simple server DEMAS in Simkit is relatively simple and straightforward. The implementation makes use of the previously built Simple Server DES and, with minimum modification, adds the MAS portion to complete the DEMAS. Table 9 list the two event codings used in Simkit to represent the MAS portion of the Simple Server DEMAS[3]. `“makeDecision()”` makes use of the average system time to gauge the happiness of the customer agent. If the customer’s system time is above the average system time of all the other customers, it will make modifications to affect the next arrival time, such as by increasing the current arrival time by 50%, decreasing it by 1 simulation time, or randomly selecting a new arrival time. Since the Simple Server DEMAS is meant to be an entry test to validate the DEMAS design concept, the MAS mental model is designed to be simple, with only five types of arrival time modifiers. In addition, the MAS decision does not keep track of the effectiveness or any reusing of the modifiers, which implies that customers may achieve worse system times. Table 10 list the parameters and settings used to run the Simple Server DEMAS.

[3] The complete Simkit code is available from the advisor to this thesis. Refer to Appendix A for the highlighted sample codes.

Table 9. Simkit Code used in Simple Server DEMAS.

Events	Simkit Implementation

Run Event	`public void doRun() {`
	`  for (int count = 0; count < noOfAgent; count ++)`
	`  waitDelay("Arrival",`
	`  agents[count].getArrivalTime(),`
	`      Priority.HIGHEST, agents[count]);`
	`}`

MAS Event	`public void doMASDecision(Agent agent) {`
	`  double ave = getAve();`
	`  makeDecision(agent, ave);`
	`  printAveSystemTime();`
	`  waitDelay("Arrival", agent.getArrivalTime(),`
	`      agent);`
	`}`

Table 10. Parameters and setting used for Both Simple Server DES and DEMAS.

Parameters and Settings	Values
Simulation Run Time	500 time units
Number of Servers	5
Number of Customers	50
Customer Inter-arrival Time	Exponential Distribution with mean of 1.7
Server Service Time	Normal Distribution with mean of 5.0 and standard deviation of 1.0

The results from the Simple Server DEMAS are favorable when compared to a Simple Server DES using only random arrival time. After 500 simulation time units, a continuous improvement in reduction of overall average system time of all customers has been observed. This improvement is the result of the MAS exhibiting complex adaptive behavior in the DES. Each customer of the MAS, when attempting to attain individual goals, adapts to the system as a whole. The DEMAS model also shows a better total number of customers being served, and a lower average number of queuing customers,

when compared to the normal DES. Figures 25 and 26 show the results of the average system time over 500 simulation time and Table 11 shows a segment of the outputs from both Simple Server Simulations.

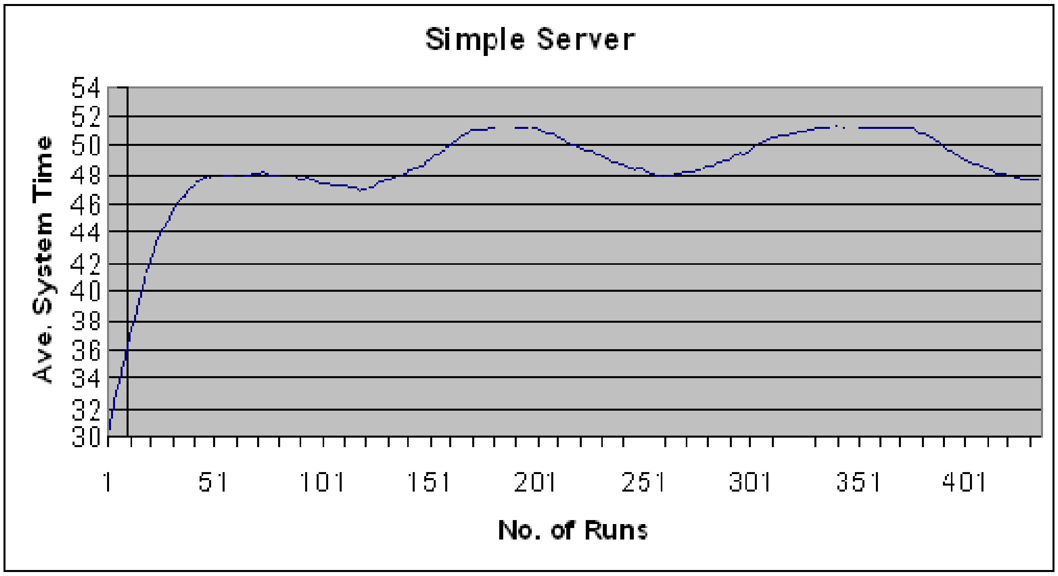


Figure 25. Average System Time over 500 runs for Simple Server DES.

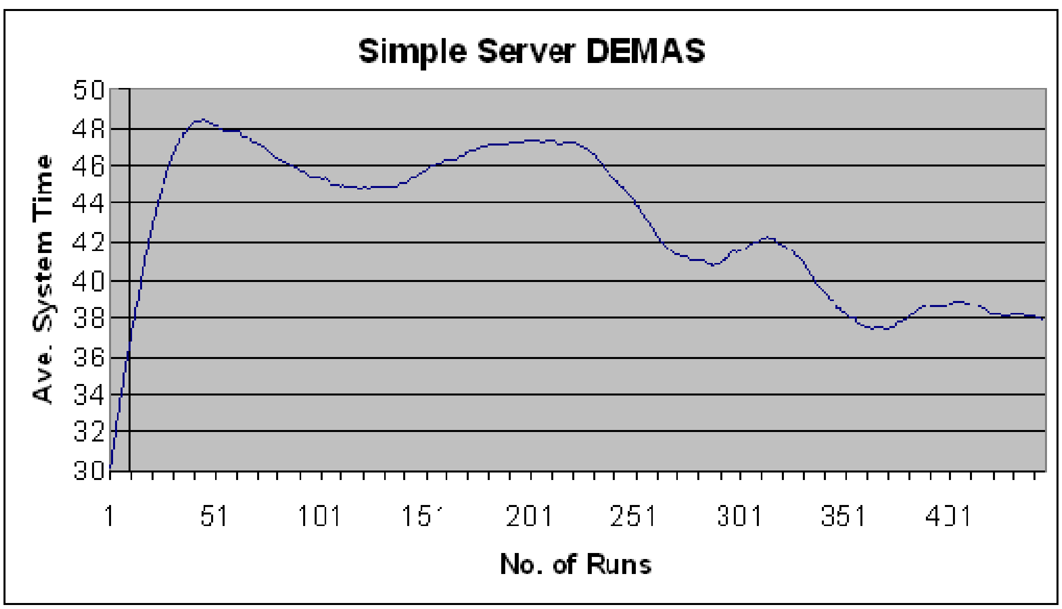


Figure 26. Average System Time over 500 runs for Simple Server DEMAS.

Table 11. Output Results Of Both Simple Server Simulation.

Simple Server DES	Simple Server DEMAS

`Simulation ended at time 500.0`
`There have been 491 customers served`
`Average number in queue 43.1086`
`Average utilization 1.0000`
	`Simulation ended at time 500.0`
	`There have been 500 customers served`
	`Average number in queue 37.7646`
	`Average utilization 1.0000`

2. El Farol Bar DEMAS

The next validation for the DEMAS design concept is centered on an implementation that starts with a given MAS. This design approach, as mentioned in the previous text, is not recommended because of the tendency to drift into a time-stepping mindset, which also nullifies the efficiency that the event-driven paradigm provides in DEMAS. The selected MAS is the El Farol Bar problem introduced by Arthur in 1994 [27]. Arthur described the El Farol Bar problem as a paradigm of complex economic systems consisting of inductive reasoning ability. In the El Farol Bar MAS model a population of agents has to decide whether to go to the "El Farol" Bar each Thursday night. The personality of the agents is crafted such that they will feel happy to go to the bar as long as it is not too crowded. Arthur tagged the attendance threshold of a crowded bar at sixty, given 100 agents in total. Before the opening hour of the bar, each agent predicts the attendance outcome of the bar, based on recent past history such as "Attendance similar to last week" or "the average attendance of the past two weeks." If most of the agents predict that bar attendance will total fewer than sixty, then most of them will turn up, crowding the bar and becoming unhappy. Conversely, if they predict that attendance will be high, they will stay home, resulting in low attendance. These sets of predictors are randomly assigned to each agent and each predictor will be assigned a weight based on the happiness outcome. Favored predictors will be reused many times, while unpopular ones will be kept out of the agents' focus.

Using the recent past attendance as the prediction models can yield a large number of reasonable and defensible models. As a result, when the agents are not sharing the knowledge of which models they are using, there is no deducible rational solution. Individually, each agent has to rely on induction behavior to determine the best suited model.

Arthur's computer simulation experiment on the El Farol Bar problem first showed the attendance at the bar oscillating in an apparently random manner. However, after some initial learning time the agents' hypotheses and mental models in use were mutually co-adapted, causing the cumulative attendance to settle near the critical sixty mark.

To model the El Farol DEMAS, a set of El Farol Java classes previously developed in a pure MAS design was used. The objective of this implementation is to demonstrate that the DEMAS design concept is able to transform an ordinary MAS design, first to improve efficiency by using discrete event drivers, and second, to complement what MAS lacks and what DES can provide. The design of the El Farol Bar problem was previously shown in Figure 22 of this chapter. The event graphs in figure 27 represent the design of the El Farol DEMAS model. The implementation makes an effort to retain the full structure of the original El Farol MAS mental model and, therefore, an additional class is introduced as part of the MAS subsystem to manage all events scheduling with the El Farol MAS mental model. Including the MAS subsystems, there are a total of three subsystems in the El Farol DEMAS. The "BarEvents" subsystem takes care of opening and closing of the bar, and also the starting and ending of queuing events. The "EnterBarEvents" subsystem, on the other hand, handles all movement in and out of the bar, and includes monitoring of agents at the queue. LEGO Listener links are used to trigger event schedules between these subsystems.

The new El Farol DEMAS added new features to be modeled. The additional features were deemed to be benefits of DES that MAS itself cannot achieve. In the original El Farol MAS, the bar was modeled as a single-day activity without consideration of the agents' arriving at different times or leaving earlier so that new agents could join in and enjoy. In the El Farol DEMAS, the different arrival time and

leaving time of each agent is modeled to provide a more realistic bar attendance situation. Similarly, instead of having sixty as the crowded threshold, the El Farol DEMAS assumes that the maximum capacity of the bar is sixty, which is again a realistic limitation to an actual bar. Lastly, the adding of the queue commonly found in many bars, such that agents who cannot enter the bar due to capacity limitation can wait at the queue.

In the El Farol DEMAS design, “BarEvents” subsystem is a straightforward implementation as compared to the “EnterBarEvents” subsystem. In the “EnterBarEvents” subsystem, customer agents arriving at the “Arrival” event may join the queue only if the queue threshold is not met. If the queue is full, the customer returns home feeling unhappy to have made the trip. This is similar to the original El Farol MAS. However, if the queue is free the customer will be allowed to join the queue and wait for an opportunity to enter the bar. This realistically models the real situation commonly seen in many pubs and bars. Customer will be allowed to enter the bar only if the bar threshold of sixty is not reached. “T_s” is the stay time preferred by the customer upon entering the bar. Both arrival time “T_a” and stay time “T_s” are uniformly distributed based on the operating hours of the bar. Customers who have joined the queue and waited to enter the bar will all be declared happy agents. This is modeled slightly different from the original El Farol MAS, which unrealistically assumed all agents to be unhappy once the bar threshold was breeched.

The Simkit implementation of the El Farol DEMAS is challenging due to the necessary steps to break up the MAS mental model from the original time-stepped environment. In the El Farol DEMAS Simkit development, the `“MAS”` class functions as the event scheduling manager to interact with the MAS mental model. Table 12 shows the “OpenBar” event call that triggers the MAS to perform the decision-making process.

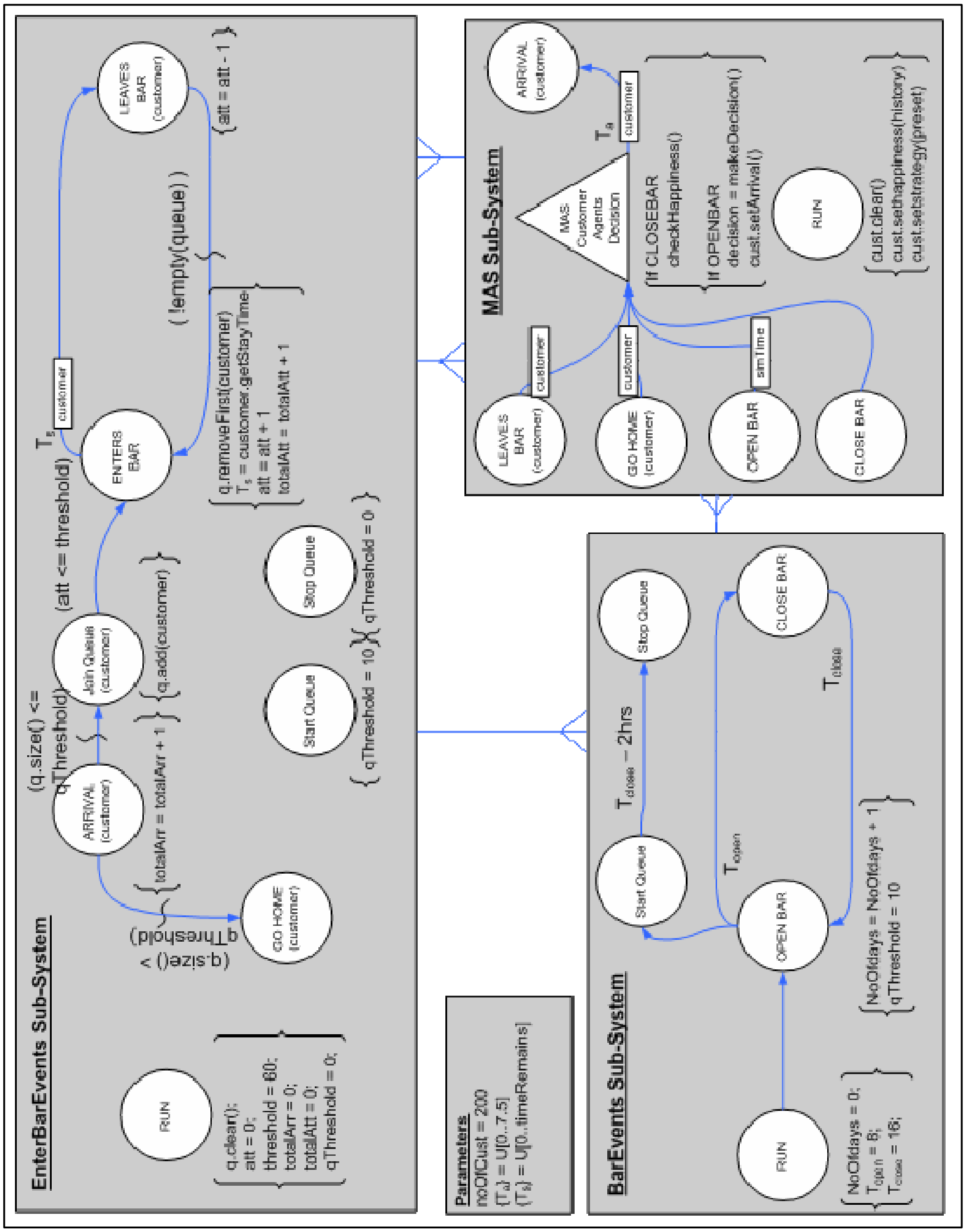


Figure 27. El Farol DEMAS Modeled with Event Graph.

The detailed design of the MAS mental model will not be illustrated in this thesis, since the focus here is to understand the potential of merging DES and MAS to form the DEMAS design concept. Nevertheless, in short summary, the MAS mental model consists of thirty-six predictors, of which eight are randomly assigned to each agent. Individual agent in turn manage its own set of eight focal predictors, increasing weight to predictors that give good prediction while reducing weight to unfavorable ones. The highest weighted predictor will be the active predictor making decision for the next move. Code samples of the implementation are provided in Appendix B for reference.

Similar to the Simple Server DEMAS, the parameters and setting to run both the El Farol MAS and El Farol DEMAS experiments are shown in Table 13.

Table 12. Simkit Code used in El Farol DEMAS.

Events	Simkit Implementation

OpenBar Event in MAS class	`public void doOpenBar() {`
	`   int attendance = 0;`
	`   RandomVariate arrTimeGenerator =`
	`      RandomVariateFactory.getInstance("Uniform",`
	`      0,7*60*60));`
	`   for (int agentCount = 0; agentCount < noOfCustomers;`
	`      agentCount ++) {`
	`      Customer cust = custArray.get(agentCount);`
	`      /** Randomly assign next arrival time. */`
	`      cust.setArrTime(arrTimeGenerator.generate());`
	`      /** Agent arriving on time, will have this`
	`       *  maximum stay time cal.`
	`       * to be 8hours - arrival time. */`
	`      cust.setMaxStayTime((8*60*60)-cust.getArrTime());`
	`      /** Decision to go or stay home is derived in`
	`       *  makeDecision function in MAS mental model`
	`       *  class. Decision is added to the attendance`
	`       *  list. */`
	`      attendance += cust.makeDecision(barThreshold);`
	`      waitDelay("Arrival", cust.getArrTime(), cust);`
	`   }`
	`   weeklyAve.add(attendance);`
	`}`

Table 13. Parameters and setting used for Both El Farol MAS and DEMAS.

Parameters and Settings	Values
Simulation Run Time	1000 weeks
Number of Customers	200
Customer Inter-arrival Time	Uniform Distribution with minimum 0 and maximum of 7 hours
Customer Stay Time (Only for DEMAS)	Uniform Distribution with minimum 0 and maximum of 8 hours – arrival time
Bar Operating hours	8 hours
Bar Close hours	6 days, 16 hours (Weekly event)
Bar Capacity Threshold	60 pax.
Queue Threshold	10 pax.
No Queue Hour	2 hours before Bar closure
Number of predictors	36
Number of focal predictors	8 per agent inclusive of 1 active predictor
Agent Personality Types	Extrovert, Introvert and Neutral

The result from the El Farol DEMAS is expected to differ from the original El Farol MAS, mainly due to the fact that the DEMAS model is the more precise and realistic El Farol Bar environment. However, the characteristics of the result should closely resemble each other. This is because they both use the same predictor sets and the same MAS mental model.

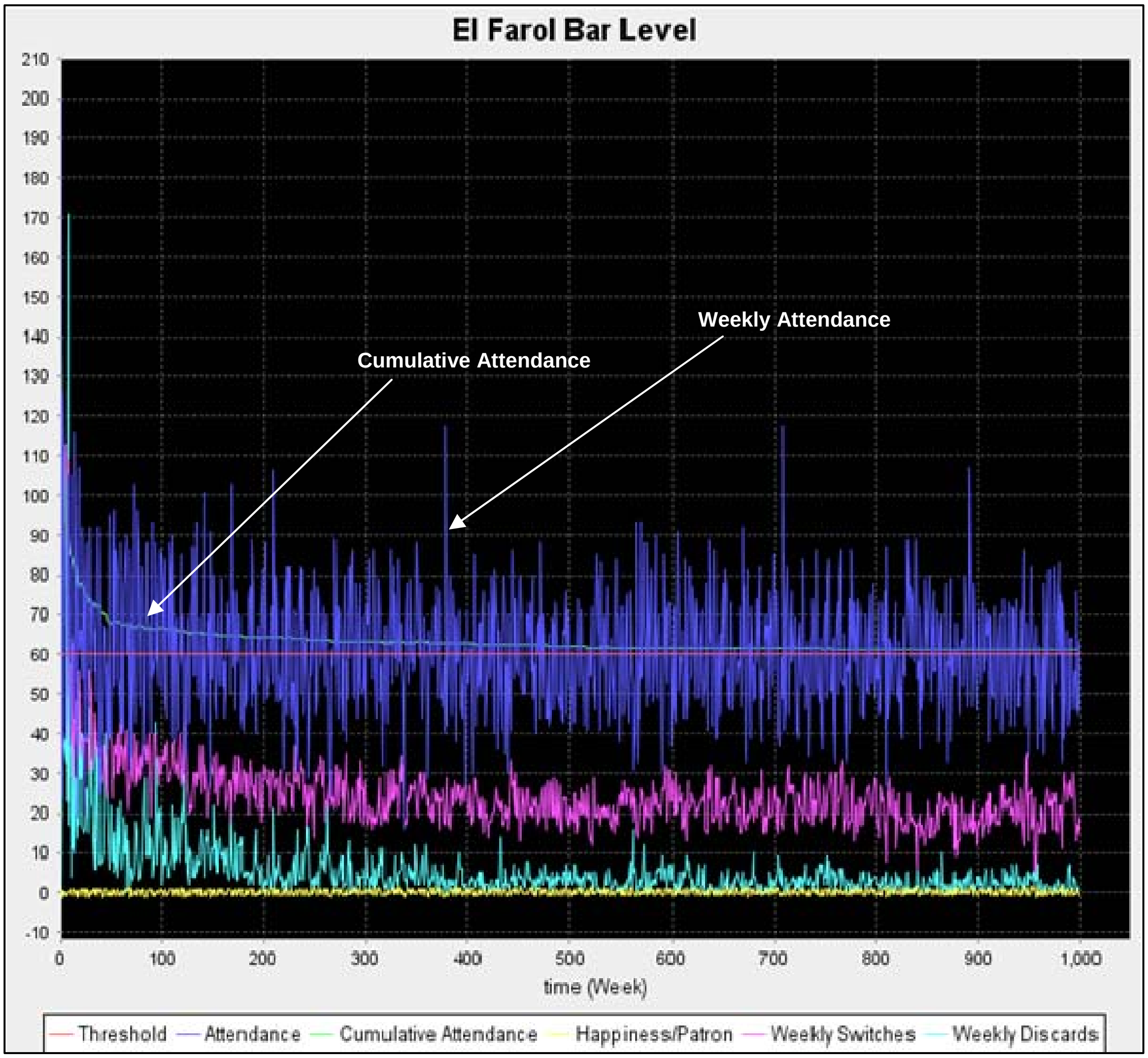


Figure 28. Weekly and Cumulative Attendance from El Farol MAS.

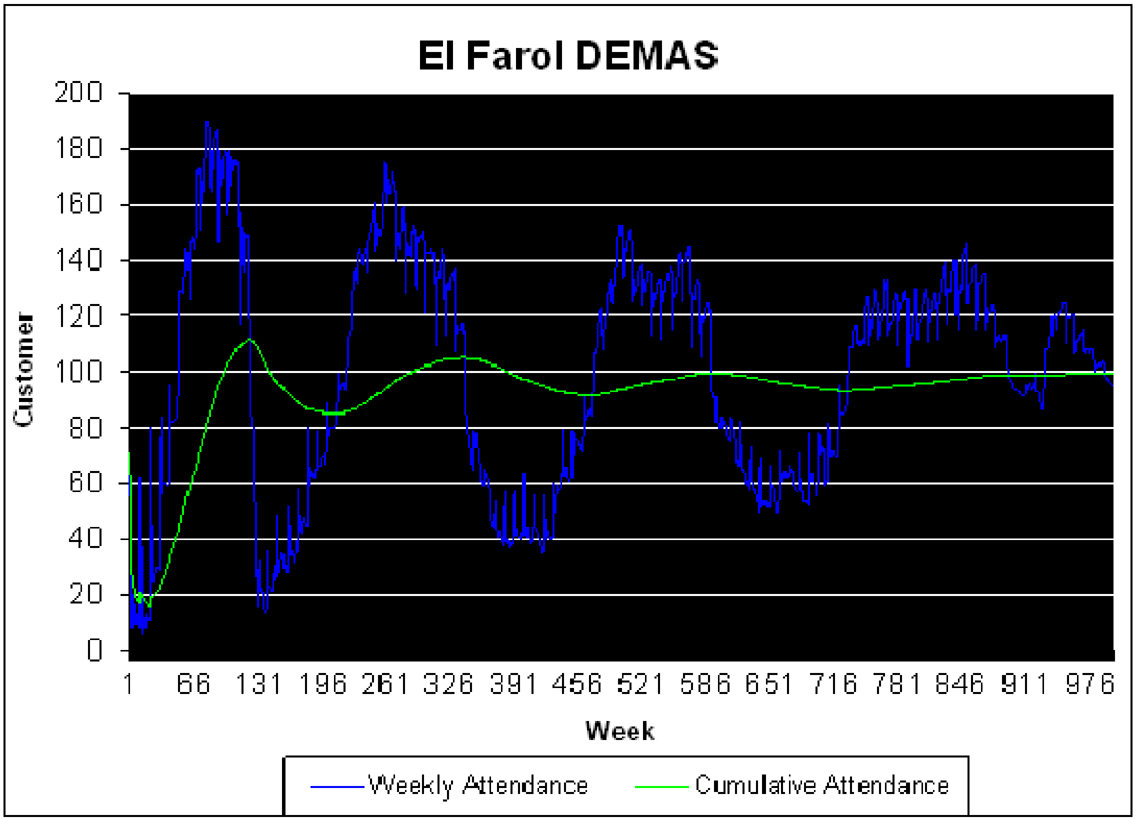


Figure 29. Weekly and Cumulative Attendance from El Farol DEMAS.

Figures 28 and 29 show the resulting attendance of the El Farol MAS and DEMAS, respectively. In Figure 28, the cumulative attendance settles closely at the bar threshold, concurring with the experiments conducted by Arthur. In the new El Farol DEMAS, the ability to model discrete time in a single day of the bar activity allowed the bar to serve more than the threshold of sixty. Figure 29 shows that the cumulative attendance of the bar stabilized at around hundred. Closer observation indicated that both graphs have similar characteristics, where the attendance oscillates around some desired point. In the MAS model, it is the bar threshold of sixty, while in the DEMAS model it is the average customers served weekly. The El Farol DEMAS can be further enhanced to model queue preferences, viability of increasing bar capacity, customer preference stay time and arrival time problems commonly needed by commercial business.

Despite the fact that the El Farol DEMAS models a slightly different aspect of the El Farol Bar problem, the design has become more realistic because of the DES model's running as the environment. For pure MAS, time stepping through every second of the Bar activities such as the modeling of the agent arriving and leaving the El Farol Bar will be computationally taxing if not impossible. Using the event-driven paradigm eliminates the computational issue, allowing bar level activities to be modeled as events take place. The DEMAS design concept provides the means to hybrid the benefits of DES and MAS for the implementation of a more realistic El Farol Bar problem.

V. IMPLEMENTING DEMAS INTO SEA11 PROJECT

A. INTRODUCTION TO SEA11 PROJECT

System Engineering and Analysis 11 (SEA11) project[4] is also termed as the Riverine Sustainment 2012 project. This project is a joint thesis involving 25 postgraduate students from the different curriculums of the NPS. The project was completed in June 2007 submitted as an M.S. graduating thesis [28].

The SEA11 project covered a relatively large research area for the U.S. Navy riverine operation, planned for year 2012. The objective is to harness presently emerging technologies in support of the expected military requirements. The scopes covered in the project consisted of four main portions, namely Force Protection, C3, Logistic Supply and Maintenance Repair. Because of the projection of advanced technologies and operation concepts that will not be available for full testing and implementation, the project group relied on simulation and mathematical models for analyses and studies. Several simulation packages were used in the process of the studies. For example, MANA was used for the study of force protection and Simkit was used to analyze the logistic supply plan.

B. INTRODUCTION TO SEA11 LOGISTIC DES

One of the main reasons to use SEA11 project as a validation to DEMAS design was because of the already implemented and validated Logistic Supply simulation developed using Simkit. It is also considered valuable to apply the DEMAS design in a real problem using realistic data. After all, the two implementation examples given in Chapter IV are considered theoretical solutions. In addition, it was because of familiarity with the SEA11 project, particularly in the development of the logistic simulation.

[4] SEA project is managed by the System Engineering and Analysis Curriculum Office of the Naval Postgraduate School. The number 11 represents the project number that runs consecutively every year within the SEA curriculum.

The SEA11 Logistic Supply simulation was built using Simkit and LEGO framework. The simulation modeled the logistic supply plan between the supply ship and the various types of riverine operating bases. Because of other operation requirements and the risk of exposure to confined area attack, the supply ship will not station near the operating base. The cycle time of the supply ship ranges from four to nine days, and it will station outside the area of operation when present. The logistic support link between the supply ship and the operating base therefore is reliant upon the smaller supply craft deployed from the operating base. The simulation objective was therefore to model the supply and scheduling plans using different supply craft combinations. The results from the simulation were used as part of the feasibility analysis, economical studies and operational planning for the overall logistic support in riverine sustainment 2012.

The SEA11 Logistic simulation modeled three types of operating base and four types of supply craft. Table 14 lists the parameters for both. The details of the various operating bases and supply crafts can be found in reference [28].

Table 14. Operating Base and Supply Craft Parameters.

Operating Base	**Capacity (ton)**	**Threshold (ton)**	**Storage (days)**
Forward Operating Base (FOB)	495	67	15
Mobile Operating Base 1 (MOB1)	529	73	15
Mobile Operating Base 2 (MOB2)	818	110	15

Supply Craft	**Unit**	**LCU-1610**	**Jim G**	**LCU-2000**	**CH-53**
Speed	kts	6 ± 2	9 ± 2	9 ± 2	100 ± 10
Weight	tons	106	330	260	10
Loading time	ton/min	2	1	2	2
Unloading time	ton/min	4	2	4	2

From initial studies, it was decided that there will only be one operating base within the area of operation, and only two supply craft operating at any one time to reduce exposure. These were the constraints applied in the simulation. The simulation was used to generate reports showing the logistical health of the operating base. Given the need to study all combinations of any two supply craft types with each type of operating base, it was determined that a total of seventy-five permutations would be required. In addition, the simulation needed to perform at least thirty runs for each of the permutations to fulfill the statistical analysis. In total, 2250 runs were generated for the SEA11 Logistic study. Figure 30 shows the screen capture of the SEA11 Logistic simulation program developed with Simkit.

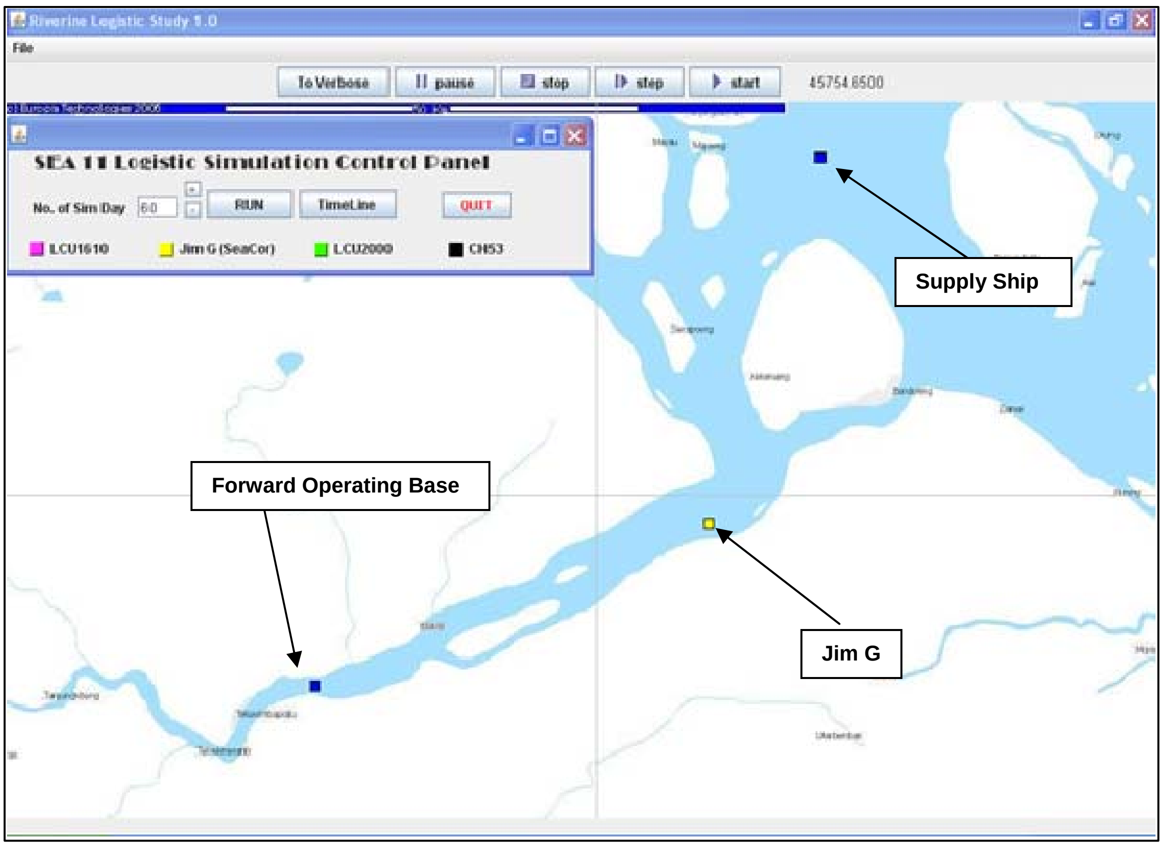


Figure 30. Screen Capture of the SEA11 Logistic DES.

After completing all 2250 sets of simulation runs, the findings from the simulation strongly suggested that the best combination of supply craft suitable for the

riverine logistic support consisted of one Jim G and one LCU2000 for all three types of operating base. The use of helicopter platforms, particularly CH-53 was found to have little operational benefit, mainly due to the high operating and maintenance cost involved.

C. DESIGNING SEA11 LOGISTIC DEMAS

The idea to implement the DEMAS design in SEA11 Logistic simulation started with the desire to reduce the simulation and analysis time. On top of the processing time to deliver the necessary data, it was also realized that the analysis process taxed available work hours with tedious data comparison. The formulae involved in the analysis revolved around time, cost and capacity - which also happen to be the goals of this SEA11 logistic support plan. If these goals can be translated into an agent's mental model, adopting the goals that minimize time and cost while maximizing utilization of the capacity, the analysis can be automated. However, because of the randomness in the supply ship's routine call and effect caused by weather conditions, the agents need to exhibit complex adaptive behavior in order to induce the correct solution.

The DES environment model used in the SEA11 Logistic DEMAS is the direct duplicate of the original SEA11 DES. Minimum modifications were applied to the model. However, since the SEA11 project was unable to model weather condition as part of the simulation, the DEMAS implementation took the extra step to implement a "Weather Agent," which affects weather conditions during operation runtime. The implemented weather agent is a naïve model centered on normally distributed foul weather occurrence and duration. More complicated weather models may be implemented in future enhancements. Figure 31 shows the overview event graph of the SEA11 Logistic DEMAS model. The light grey subsystems are the same as the original simulation, while the dark gray subsystems are the added DEMAS subsystems. Since the "SeaSupplyCraftScheduler" subsystem is designed as the scheduler to deploy supply craft, the "SupplyCraftAgent" subsystem interacts directly with it. The "WeatherAgent" subsystem has a listener link from the "SeaSupplyCraftManager" subsystem because "SeaSupplyCraftManager" dictates the speed during the movement of the supply craft and also the capacity during loading and unloading before any movement.

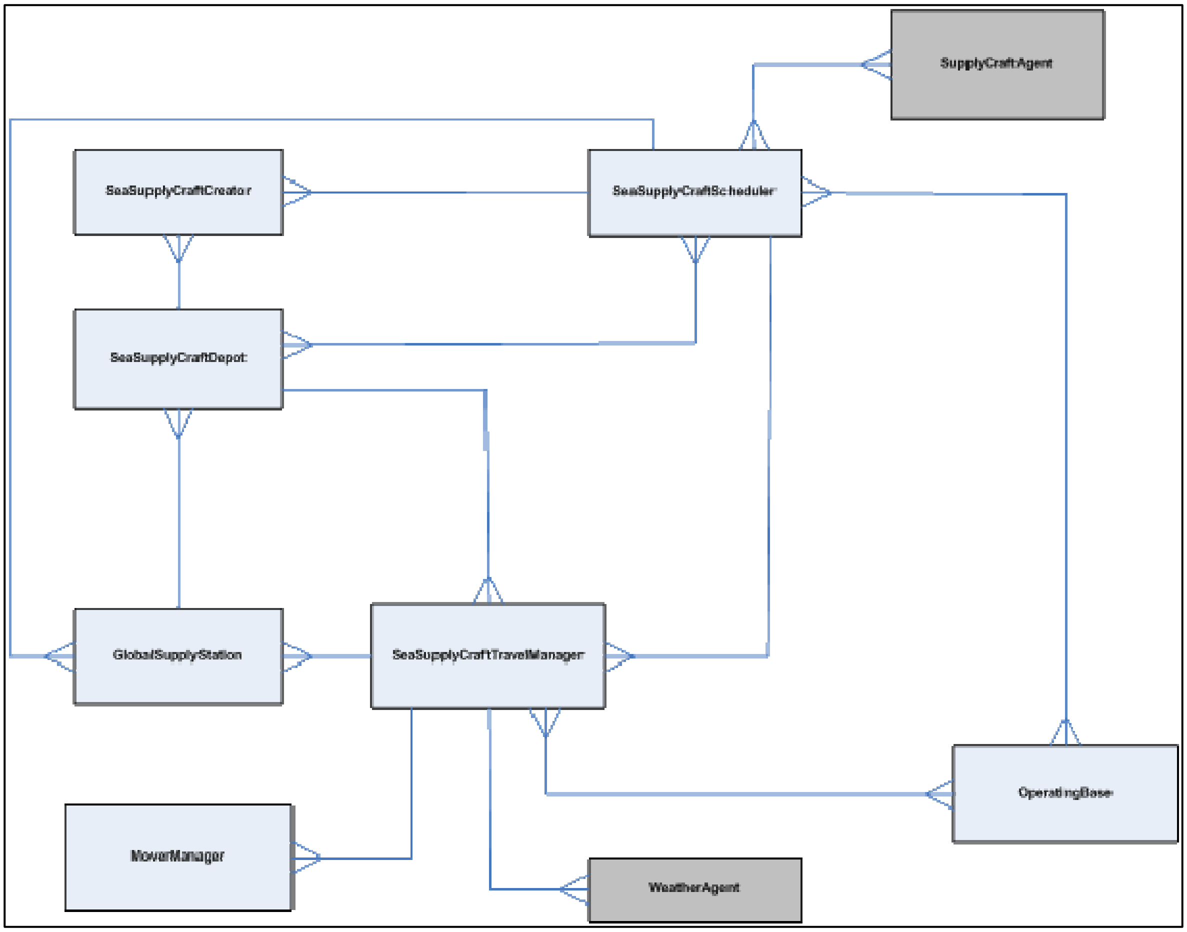


Figure 31. SEA11 Logistic DEMAS Event Graph Modules.

Figure 32 zooms into the event graph's design of the "SeaSupplyCraftScheduler" subsystem. The light grey events are originated from the SEA11 Logistic simulator, and while the dark grey events indicate the existence of listener relationships with the "SupplyCraftAgent" MAS subsystem, only the "AgentDecision" event is a new add-on. In the design, the "PreDeploy" event schedules the "AgentDecision" event to trigger the MAS, and this in turn makes the decision to schedule which supply craft to be deployed. The "Absence" and "Presence" events representing the main supply ship, when scheduled, will trigger their respective perception information to the MAS, and this is likewise when the supply craft returns to the docking station where the "SSC Returns" event is scheduled.

Figure 33 shows the pictorial design of the supply craft MAS mental model. In this mental model, a set of thirty voters are assigned to each supply craft (two for each type of supply craft). Each voter is randomly assigned a set of three models from the total of fifteen generated from data concerning the supply craft's capability and the presented operation status. Within these three assigned models, each voter continuously evaluates the correctness when input data is perceived from discrete events state changes. Evaluation is done by changing the weight, such that the highest weighted model will be the active decision model. This is similar, in concept, to the El Farol MAS design. When supply demand is requested from the operating base, the MAS will be triggered to consolidate voting from the voters of the available supply craft within the depot. From the decision process, the sea supply craft with the highest vote for deployment will be selected and scheduled for an actual mission by the "SeaSupplyCraftScheduler" subsystem.

The weight and correctness in the decision models are updated at various event schedules within the operation timeframe. Correctness of a decision model is based on improvement measured at each deployment. In other words, if the decision model suggested for deployment, and the deployment itself, yield improvement in the delivery time, cost and capacity, then the decision model will be deemed correct and vice versa. The majority of the data, such as overall deployment time and capacity requested, are fed into the input suite of the MAS when the supply ship arrives or departs from the area of operation. Other minor components such as average deployment speed and average capacity will be received when a supply craft returns from mission to docking station.

The Simkit development of the new Logistic DEMAS requires the additional MAS classes to be linked into the original Simkit simulation. Using LEGO framework allows encapsulation and modularity, which in turn minimizes modification required to establish these links. Simkit code samples for SEA11 Logistic DEMAS are provided in Appendix C for reference. The simulation parameters are shown in Table 15 (page 65).

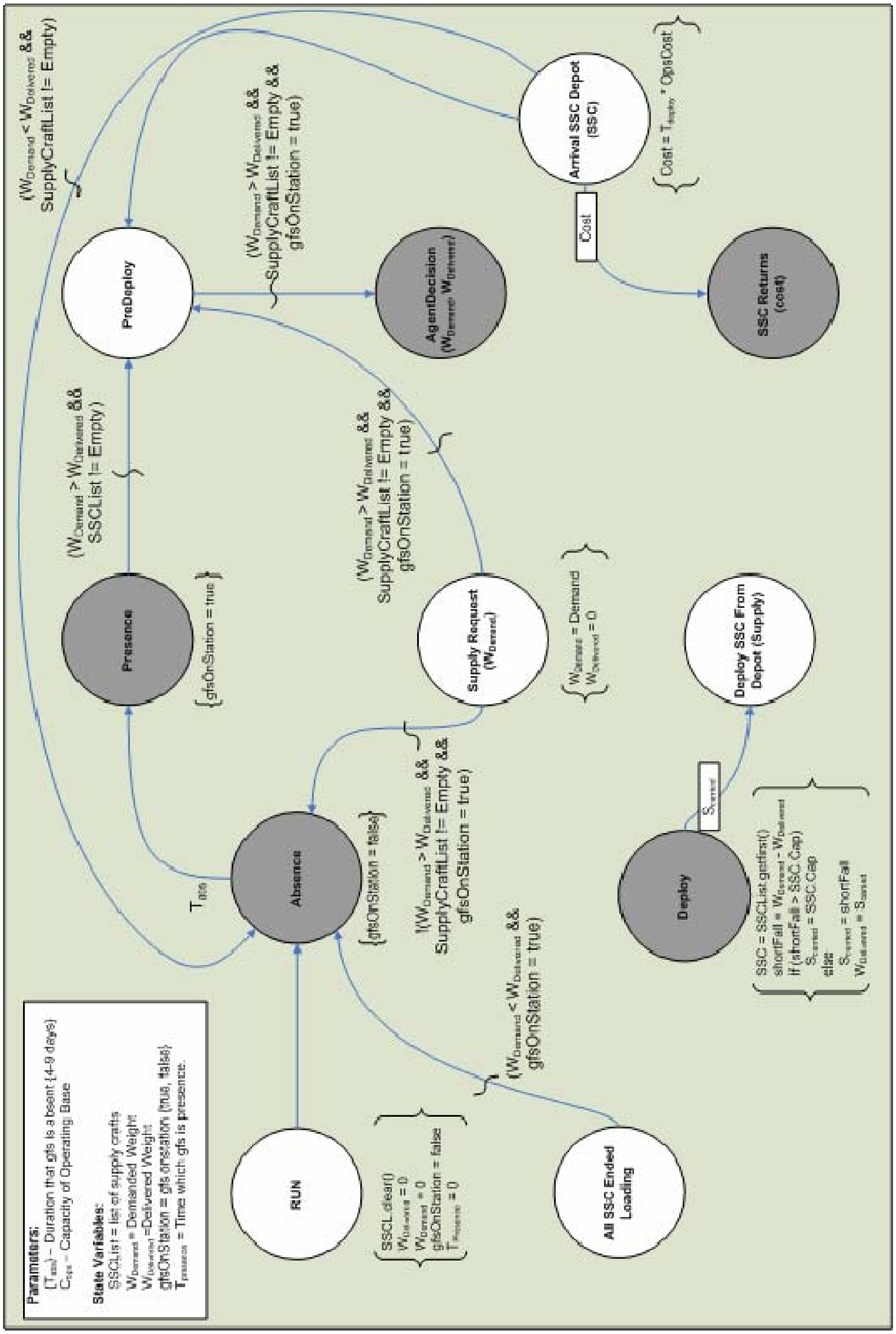


Figure 32. Event Graph for “SeaSupplyCraftScheduler” Subsystem.

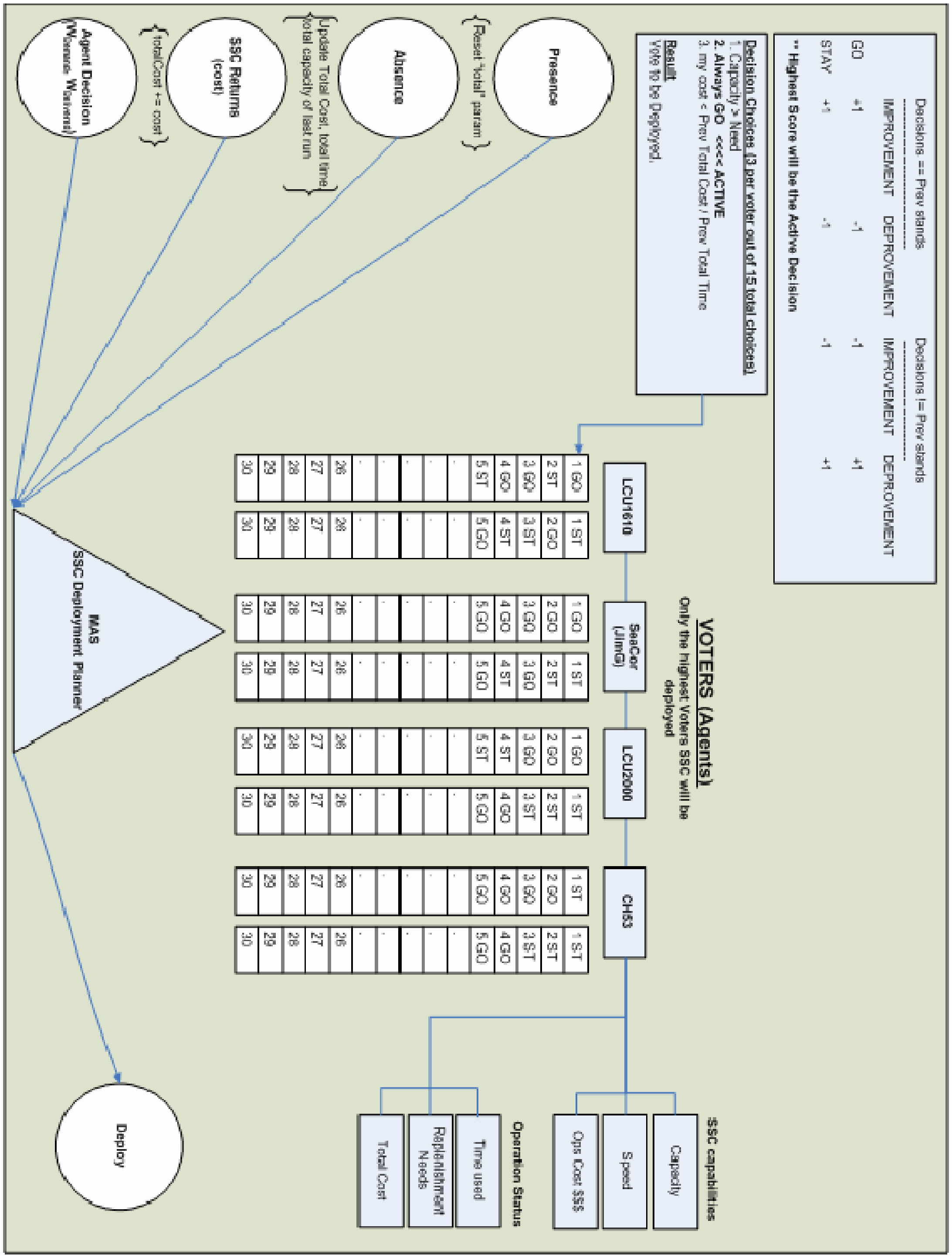


Figure 33. SEA11 Logistic DEMAS Event Graph for MAS "AgentDecision" Module.

Table 15. Parameters and Settings used for SEA11 Logistic DEMAS.

Parameters and Settings	Values
Simulation Run Time	180 days
Number of Agents (voters)	30 x 8 = 240
Supply Ship Cycle Time	Triangle distribution centered at 6 days and ranges from 3 to 9 days.
Number of Decision Models	15 decision model, 3 randomly assigned to each agent
Foul Weather Occurrence	Normal distribution around 10 days with standard deviation of 2 days
Foul Weather Duration	Normal distribution around 1 day with standard deviation of 3 hours

D. RESULT AND ANALYSIS

By adding intelligence into the SEA11 Logistic DES, it is hoped that the agents will induce and adapt to the overall environmental and operational goals. The resulting deployment plans and profiles would likely match the findings from the SEA11 analysis. Figure 34 shows the Logistic DEMAS deployment time chart for the supply craft. A full solid line across indicates no deployment throughout the simulation. Pockets of lines indicate deployment duration of the supply craft. Figure 35 shows a clearer blown-up portion of the deployment time chart. Foul weather condition is shown in the first row across, while the remaining rows represent each of the supply craft.

From the deployment time line chart, it is clearly indicative that the MAS decision continuously chooses to deploy one Jim G (Seacor) and one LCU 2000 at different intervals. A CH-53 was deployed only once throughout the simulation, on approximately 29 April. It is also observed that both LCU 2000s were deployed but at a different time frame. The result further indicated that a single Jim G or a single LCU 2000 supply craft

is sufficient to sustain the operational profile of the operating base for a period up to 180 days. These results and analysis match the findings of the SEA11 project, therefore validating the ability of the new DEMAS system to perform logistic planning with human-like considerations and objectives. More importantly, the DEMAS design eliminated the need to run the simulation 2250 times. Instead, for a complete coverage of modeling all three operating base types, and to obtain thirty runs per base to fulfill the statistical requirement, the SEA 11 Logistic DEMAS only requires ninety simulation runs.

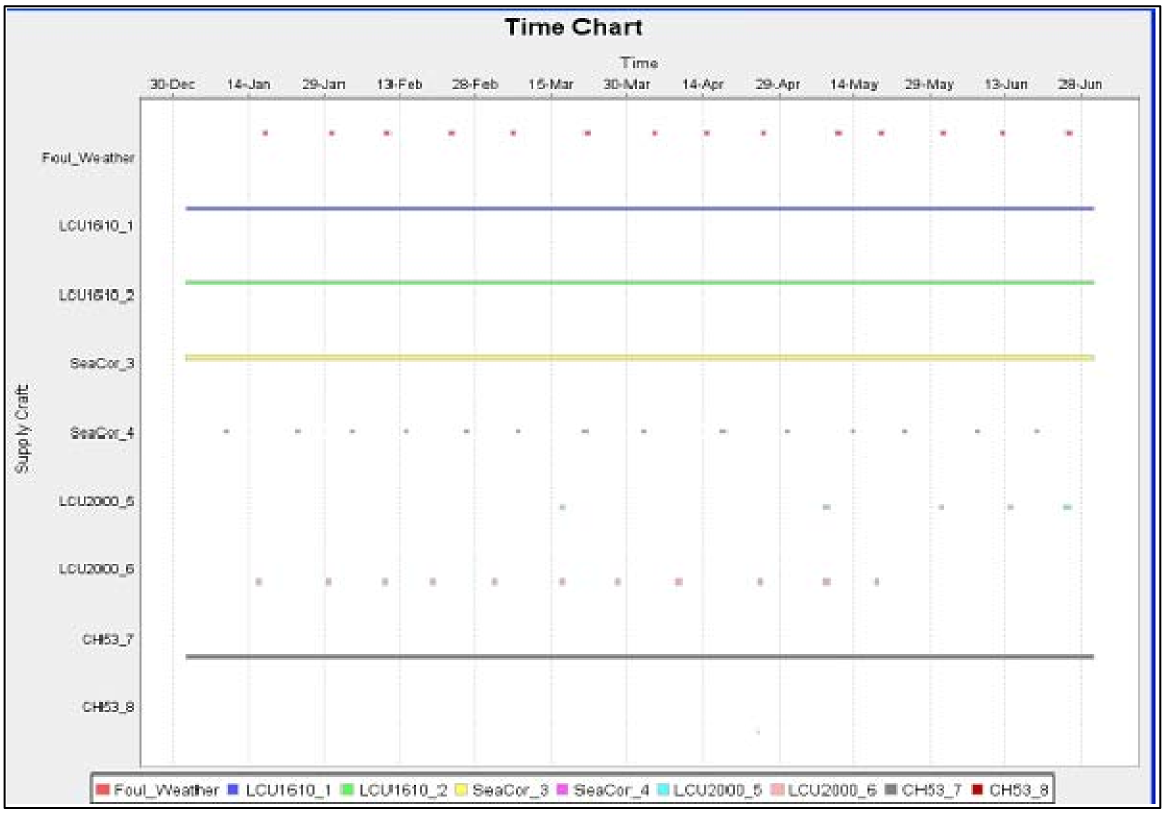


Figure 34. SEA11 Logistic DEMAS Result.

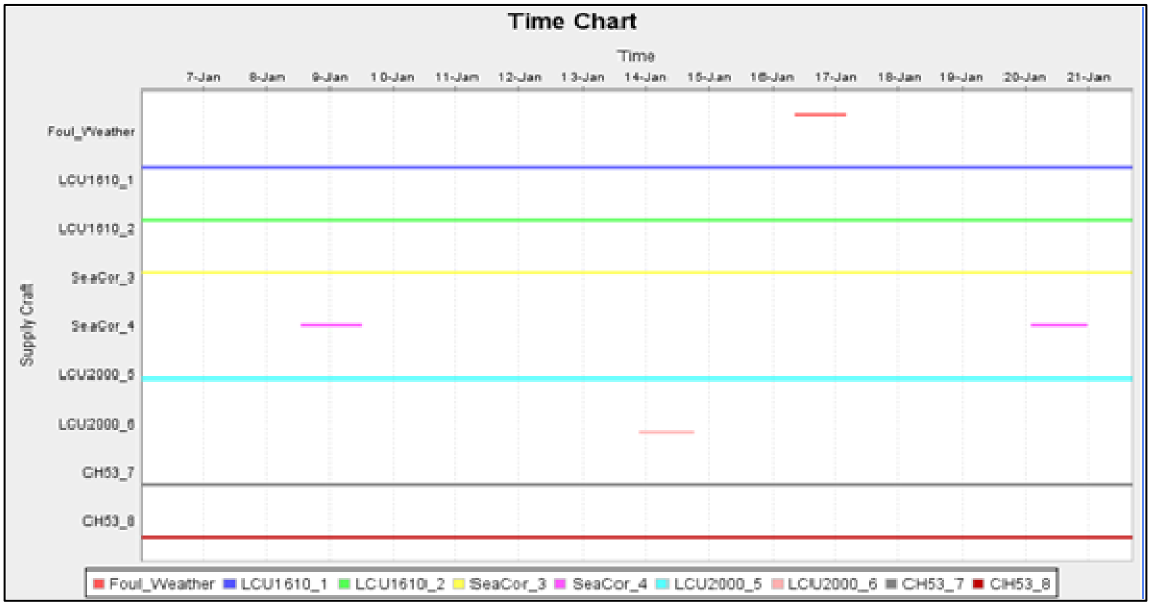


Figure 35. Zoom In View Of SEA11 Logistic DEMAS Result.

To further challenge the DEMAS model, the operating base capacity and consumption rate were increased ten times from the original value. This will render the replenishment capacity of a single Jim G or LCU 2000 insufficient in fulfilling the supply request. The purpose of this diversion is to verify that the agent mental model in the Logistic DEMAS is able to adapt to harsher demand. Figure 36 shows the deployment time line chart, highlighting that multiple supply craft must be deployed at each supply request mission, in order to fully replenish the operating base.

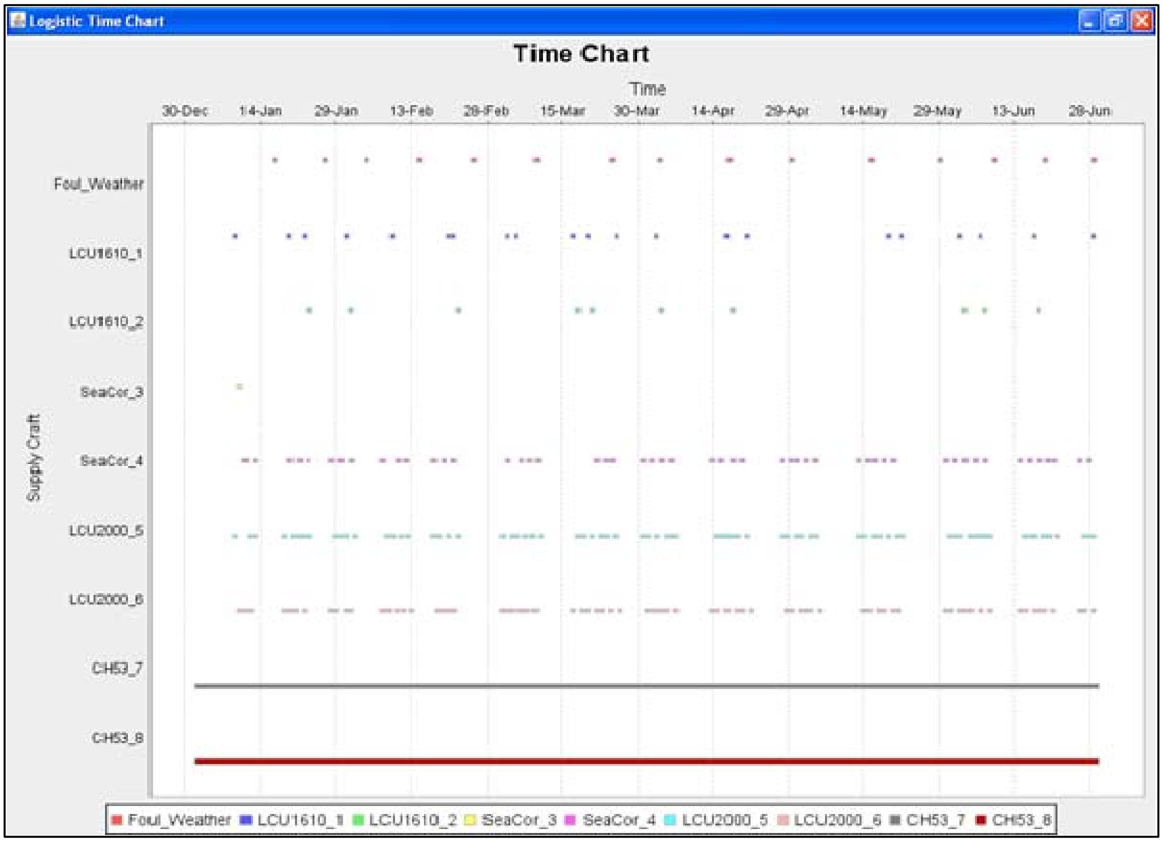


Figure 36. SEA11 Logistic DEMAS Result (For Operating Base with Increased Capacity And Consumption Rate).

Figure 37 display a blown up portion of the chart. Jim G and LCU 2000 were both equally popular for deployment as compared to LCU 1610 and CH-53. The time line chart displayed a sparse deployment profile for both the LCU 1610. The graph also shows no deployment for either of the CH-53 helicopters, which is probably due to their limited replenishing capacity with respect to increase in capacity and consumption rate of the operating base. It was observed that Jim G supply craft 3 was only deployed once during the 180-day operation, which is possibly due to undesirable random assignment of the decision model within the voter agents. The deployment time line chart also reveals randomness in the deployment profile, suggesting the adaptive behavior of the MAS attempting to adapt to the operational needs.

Figure 37. Zoom in View of the SEA11 Logistic DEMAS Result (For Operating Base with Increased Capacity and Consumption Rate).

The analysis continues to suggest that Jim G and LCU 2000 are the best combination in most cases, even when deployed as a pair. CH-53, due to the limited capacity and high operating cost, is not feasible given such operation requirement.

The integration of MAS into the SEA11 Logistic DES using the DEMAS design concept was successful and has proven to be beneficial. Future enhancement can include the addition of a more complex weather agent model that will realistically put the simulation to test. Further analysis can also be obtained from analyzing the adaptive behavior of the MAS for future logistic support planning. This chapter concludes the validation of the DEMAS design concept in merging MAS into DES.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

The benefits of merging DES and MAS were apparent and have been supported by other researches, such as SWARM, SPADE and AOR metamodel. A successfully merged simulation will exhibit intelligence, by means of agents with human-like adaptive behavior, interacting with an efficient discrete event world, advancing using the event-driven paradigm.

A new design concept, termed the Discrete Event Multi Agent Simulation or DEMAS, was proposed as the designing approach to merge Multi Agent System and Discrete Event Simulation. The concept approaches the merger by first defining the DES environment as independent subsystems, followed by defining the input and output suite of the MAS. Lastly, the design is completed by linking the interaction and triggers between the DES subsystems and the MAS. The DEMAS design uses Event Graph Methodology as the designing tool because of its simple, yet powerful representation of DES. Listener Event Graph Objects (LEGO) framework is applied into the event graph in order to support modularity design for DEMAS.

To validate the feasibility and usability of the DEMAS design concept, two implementations were introduced. First was the Simple Server DEMAS, which resulted in agents' adapting and improving the overall system time. Likewise, in the second implementation the El Farol DEMAS validated that a more efficient and realistic bar environment can be modeled without losing the adaptive ability of the MAS. The DEMAS development uses Simkit as the development library due to close correlation between Simkit and event graph. In addition, Simkit supports the LEGO framework.

The final implementation involves application of the DEMAS design concept in a larger, more realistic simulation model consisting of real data analysis. The SEA11 Logistic DES was added with MAS to produce the SEA11 Logistic DEMAS. The DEMAS design successfully allowed human-like planning consideration to be added to the simulation, with a much lesser simulation runtime requirement.

The DEMAS design concept has been validated with the ability to retain the benefits of both DES and MAS combined. In addition, the DEMAS design concept, which uses Event Graph, LEGO framework and Simkit libraries, provided added advantages such as simplicity in representation and modularity design that will assist future researches and development.

Future researches on DEMAS should look into topics such as autonomous new event creations. This is a higher level of autonomous agent, where the agent no longer relies on pre-defined events, but is also able to create new events to interact with the simulated world.

APPENDIX A. SIMPLE SERVER DEMAS

A. SIMPLE SERVER DISCRETE EVENT CODE SAMPLE

```
/**
 * EndService event increment the number of available servers
 * and the total number of customer being served. It will schedule
 * MASDecision upon completion of current service. MASDecision will
 * capture the total system time, compute an average and update the
 * new interarrival time of the customer for the next arrival. It
 * will schedule another StartService event with no delay if there are
 * customers waiting in the queue (Q > 0).
 */
public void doEndService(Agent agent) {
  double newSystemTime = agent.getElapedTime();
  int oldNumberAvailableServers = getNumberAvailableServers();
  int oldNumberServed = getNumberServed();
  numberAvailableServers = numberAvailableServers + 1;
  numberServed = numberServed + 1;
  firePropertyChange("numberAvailableServers",oldNumberAvailableServers,
      getNumberAvailableServers());
  firePropertyChange("numberServed", oldNumberServed,getNumberServed());
  if (getNumberInQueue() > 0) {
      waitDelay("StartService", 0.0);
  }
  agent.setSystemTime(newSystemTime);
  waitDelay("MASDecision", 0.0, agent);
}
```

Figure 38. doEndService Code Sample in SimpleServer Class.

B. SIMPLE SERVER MAS MENTAL MODEL CODE SAMPLES

```
/**
  * MASDecision event accept 1 argument namely the Agent entity. This is
  * to tell the system that the agent(customer) has completed being
  * served by the Server and is ready to decide on the next arrival
  * time.
  */
public void doMASDecision(Agent agent) {
  double ave = getAve();
  makeDecision(agent, ave);
  waitDelay("Arrival", agent.getArrivalTime(), agent);
}
```

Figure 39. doMASDecision Code Sample in AgentProcess Class.

```java
/**
  * makeDecision function make decision to affect the new arrival time
  * as long as the System time for the agent is above the average System
  * Time of all the agents.
  * @param agent The agent entity.
  * @param ave The average of all the agents' system time.
  */
public void makeDecision(Agent agent, double ave) {
  if (agent.getSystemTime() > ave) {
      String distributionSer = "Uniform";
      double alpha = 0;
      double beta = 5;
      RandomVariate decisionRnd =
          RandomVariateFactory.getInstance(distributionSer,alpha,beta);

      int newStrategy = (int) decisionRnd.generate();

      if (newStrategy == 0)
          agent.setArrivalTime(agent.getArrivalTime() * 1.5f);
      else if (newStrategy == 1)
          agent.setArrivalTime(agent.getArrivalTime()* 0.5f);
      else if (newStrategy == 2)
          agent.setArrivalTime(agent.getArrivalTime() + 1);
      else if (newStrategy == 3) {
          agent.setArrivalTime(agent.getArrivalTime() - 1);
      if (agent.getArrivalTime() < 0d)
          agent.setArrivalTime(ave);
  }
  else if (newStrategy == 4) {
      distributionSer = "Exponential";
      double mean = 1.7;
      RandomVariate arrTime =
          RandomVariateFactory.getInstance(distributionSer, mean);
      agent.setArrivalTime(arrTime.generate());
  }
}
```

Figure 40. makeDecision Code Sample in AgentProcess Class.

APPENDIX B. EL FAROL BAR DEMAS

A. EL FAROL DISCRETE EVENT CODE SAMPLES

```
/**
 * When customers arrived, Arrival event will be scheduled.
 * Increment the total number of arrival only if the customer is meant
 * to enter the bar. This is the way to the happiness coding of MAS to
 * find out if the customer's decision is correct based on the decision
 * made. Only by "forcing" the customer to arrive that the actual queue
 * length and attendance in the bar can be determine to judge the
 * correctness of the decision.
 * Customer joins the queue if the decision is to go to the bar.
 * Customer goes home if the decision is not to go to the bar or when
 * the queue is full.
 */
public void doArrival(Customer cust) {
   if (cust.getLastDecision() == 1) {
       totalArrival = totalArrival + 1;
       if (queue.size() < qThreshold)
           waitDelay("JoinQueue", 0.0, cust);
       else
           waitDelay("GoHome", 0.0, cust);
   }
   else
       waitDelay("GoHome", 0.0, cust);
}
```

Figure 41. doArrival Code Sample in EnterBarEvents Class.

```
/**
 * JoinQueue will time stamp the customer, add the customer to the queue
 * and when the bar is below the threshold, schedules the customer to
 * enters the bar.
 */
public void doJoinQueue(Customer cust) {
   LinkedList<Customer> oldQueue = getQueue();
   cust.stampTime();
   queue.add(cust);
   firePropertyChange("queue", oldQueue, getQueue());
   if (currAtt < threshold)
       waitDelay("EntersBar", 0.0, Priority.HIGH);
}
```

Figure 42. doJoinQueue Code Sample in EnterBarEvents Class.

```
/**
 * EntersBar simple increase the total number of customer served and
 * determine the time which the customer will leave the bar.
 */
public void doEntersBar() {
   LinkedList<Customer> oldQueue = getQueue();
   Customer cust = queue.removeFirst();
   firePropertyChange("queue", oldQueue, getQueue());
   totalAtt = totalAtt + 1;
```

```
   int oldCurrAtt = getCurrAtt();
   currAtt = currAtt + 1;
   firePropertyChange("currAtt", oldCurrAtt, getCurrAtt());
   double oldTimeInQueue = getTimeInQueue();
   timeInQueue += cust.getElapedTime();
   firePropertyChange("timeInQueue", oldTimeInQueue, getTimeInQueue());
   // New Stay Time affected by the queuing time.
   double newStayTime = cust.getMaxStayTime() - cust.getElapedTime();
   RandomVariate stayTimeGenerator = RandomVariateFactory.getInstance
       ("Uniform", 0, newStayTime);
   waitDelay("LeavesBar", stayTimeGenerator.generate(), cust);
}
```

Figure 43. doEntersBar Code Sample in EnterBarEvents Class.

```
/**
 * Customer leaving the bar is a happy customer. LeavesBar schedules
 * the next EntersBar if the waiting queue is not empty.
 */
public void doLeavesBar(Customer cust) {
   int oldCurrAtt = getCurrAtt();
   currAtt = currAtt - 1;
   firePropertyChange("currAtt", oldCurrAtt, getCurrAtt());
   if (!queue.isEmpty())
       waitDelay("EntersBar", 0.0, Priority.HIGH);

}
```

Figure 44. doLeavesBar Code Sample in EnterBarEvents Class.

B. EL FAROL MAS MENTAL MODEL CODE SAMPLES

```
/**
 * Constructor for MAS. Setting the starting week count because of the
 * need to include some history for MAS decision (Preprocessing).
 * Define the number of customer and the number of Focal Predictors for
 * each customer. unbalancedPersonality is only used for MV4015 project.
 */
public MAS(int barThreshold) {
    this.barThreshold = barThreshold;
    startWeek = 7; // Pre history for 7 weeks.
    predictorThreshold = -10; // when will predictor be discarded?
    noOfCustomers = 200;
    noOfFocalPredictors = 8;
    unbalancedPersonality = false;
    // Generate a set of fake history randomly. One time only.
    weeklyAve = new ArrayList<Integer>();
    preWeekHistoryGenenrator(startWeek);
    // Set up the predictors to have some preset value for past 7 days.
    predictors = new ElFarolPredictorsManager(predictorThreshold);
    predictors.updatePredictor(getWeeklyAve());
    // Create a number of customers.
    custArray = new ArrayList<Customer>();
    for (int agentCount = 0; agentCount < noOfCustomers; agentCount ++) {
       Customer cust = new
       Customer(noOfFocalPredictors, predictors,
       unbalancedPersonality);
       custArray.add(cust);
    }
```

```
}
```

Figure 45. MAS Constructor Code Sample in MAS Class.

```
/**
 * Linked to the customer (MAS) decision maker function.
 * OpenBar schedules decision making process for all customer. The
 * decision to go to the bar will be determine by the value of the focal
 * predictors assigned to each customer. makeDecision uses the El Farol
 * Project in MV4015 Summer 2007.
 */
public void doOpenBar() {
   int attendance = 0;
   RandomVariate arrTimeGenerator =
      RandomVariateFactory.getInstance("Uniform", 0, (7*60*60));
   for (int agentCount = 0; agentCount < noOfCustomers; agentCount ++) {
      Customer cust = custArray.get(agentCount);
      // Randomly assign arrival time to agent.
      cust.setArrTime(arrTimeGenerator.generate());
      // Agent arriving on time, will have this maximum stay time cal.
      // to be 8hours - arrival time.
      cust.setMaxStayTime((8.0 * 60 * 60) - cust.getArrTime());
      // Decision to go or stay home is derived in makeDecision
      // function in MAS mental model class. Decision is added to
      // the attendance list.
      attendance += cust.makeDecision(barThreshold);
      waitDelay("Arrival", cust.getArrTime(), cust);
   }
   // Keeping track of the weekly attendance.
   weeklyAve.add(attendance);
}
```

Figure 46. doOpenBar Code Sample in MAS Class.

```
/**
 * When the GoHome Event is scheduled at the EnterBarEvents, the
 * correctness of the last decision is weighted. For customer whom has
 * decided to go to the bar but could not join the queue, the decision
 * gets an incorrect weight. OTOH, customer whom has decided not to go
 * to the bar but got rejected from joining the queue, will get a
 * correct weight added to the decision.
 */
public void doGoHome(Customer cust) {
   // If the agent decision was not to go (0), then by not being able
   // to join the queue is the right decision initially.
   if (cust.getLastDecision() == 0)
      cust.addCorrectness(1); // Right decision made.
   else
      cust.addCorrectness(0); // Wrong decision made.
}
```

Figure 47. doGoHome Code Sample in MAS Class.

```java
/**
 * Agent leaving the bar after successful entering has decision with
 * correct weight.
 */
public void doLeavesBar(Customer cust) {
   //Agent leaving the bar as a happy agent.
   cust.addCorrectness(1);
}
```

Figure 48. doLeaveBar Code Sample in MAS Class.

```java
/**
 * CloseBar event trigger the MAS to compute and updates all the
 * predictors' correctness.
 */
public void doCloseBar() {
   // Update all correctness in the predictor at close time.
   predictors.updateCorrectness(barThreshold, weeklyAve.get(weeklyAve.
      size()-1));
   // Update all agent focals predictors at close time.
   for (int agentCount = 0; agentCount < noOfCustomers; agentCount ++)
      custArray.get(agentCount).updateFocals();
   System.out.println();
   printHappiness();
}
```

Figure 49. doCloseBar Code Sample in MAS Class.

APPENDIX C. SEA11 LOGISTIC DEMAS

A. SEA11 LOGISTIC DISCRETE EVENT CODE SAMPLES

```
/**
 * preDeploy event schedules the AgentDecision so that the deployment
 * will be based on the highest voted SSC.
 * The new SSCList will contain the ranked SSC to be deployed.
 */
public void doPreDeploy(){
    if((deploymentCount > 0) && this.getGfsOnStation()){
       if(this.weightDemand > this.weightDelivered) {
          waitDelay("AgentDecision",0.0,
              getWeightDemand(),getWeightDelivered());
       }
    }
}
```

Figure 50. doPreDeploy Code Sample in SupplyCraftScheduler Class.

```
/**
 * ClearWeather event schedules the ChangeWeather event after a delay of
 * foulWeatherRV time. It also pass in a effect of 0.8 (20% reduction)
 * to the SSC capability. foulWeatherRV is a simple Normal distribution
 * randomness
 */
public void doClearWeather (){
   if (startOfWeatherChange != 0) {
      endOfWeatherChange = eventList.getSimTime();
      timeChart.addSubTask(foulWeatherTime, startOfWeatherChange,
         endOfWeatherChange);
   }
   waitDelay("ChangeWeather", foulWeatherRV.generate(),
      Priority.HIGHEST, 0.8);
}
```

Figure 51. doClearWeather Code Sample in WeatherAgent Class.

```
/**
 * ChangeWeather event causes foul weather in the simulation. It takes
 * in a parameter representing the effect towards the SSC, and it
 * schedules the ClearWeather event after a delay of clearRV. ClearRV is
 * a simple Normal distribution randomness.
 * @param weatherEffect The effect to the speed, loading time and
 * unloading time of the SSC.
 */
public void doChangeWeather (double weatherEffect){
   startOfWeatherChange = eventList.getSimTime();
   waitDelay("ClearWeather", clearRV.generate(),Priority.HIGHEST);
}
```

Figure 52. doChangeWeather Code Sample in WeatherAgent Class.

B. SEA11 LOGISTIC MAS MENTAL MODEL CODE SAMPLES

```
/**
 * AgentDecision prepare and call for the voting to SSC deployment.
 * The highest voted SSC will be placed on the front of the linked-list
 * for deployment. It schedules "DEPLOY" upon completion of the voting.
 */
public void doAgentDecision(double weightDemand,double weightDelivered){

   // Update new needs and aveSpeed thus far.
   double need = weightDemand - weightDelivered;

   // 1. Determine if the last result yield improvement?
   // 2. Ranked Voter IDs based on voting and parse to ranking array.
   // 3. Sort SSC List according to Ranking array.
   // 4. Update the winner list and also the voters' last decision.

   boolean improvement = getDeploymentResult(need, totalCost);
   int[] ranking = rankVoters(need, improvement);
   sortSSCList(ranking);
   updateWinner(ranking[0]);
   // First deployment, just add the craft's speed in. Else add and ave
   // it.
   if (aveSpeed == 0)
      aveSpeed = seaSupplyCraftList.get(0).getSpeed();
   else
      aveSpeed = (aveSpeed + seaSupplyCraftList.get(0).getSpeed()) / 2;
   // Deploy the SSC for duty. */
   waitDelay("Deploy", 0.0, Priority.HIGHEST);
}
```

Figure 53. doAgentDecision Code Sample in SupplyCraftAgent Class.

```
/**
 * Adding the result of the need and cost into the ArrayList
 * winnerResult. Need is set at 80% importance while cost at 20%
 * influence. For improvement to be true, the reduction must be 10% or
 * better.
 * @param need The needed capacity still on request.
 * @param cost The cost of the transfer thus far.
 * @return A boolean indicating if there's improvement.
 * (true=improvement)
 */
private boolean getDeploymentResult(double need, double cost) {
   boolean improvement = false;
   double result = (need * 0.8) + (cost * 0.2);
   // At least 10% improvement then previous result.
   if (winnerResult.size() != 0) { // If not the first result.
      if (result < (0.9 * winnerResult.get(winnerResult.size()-1)))
         improvement = true;
   }
   winnerResult.add(result);
   return improvement;
}
```

Figure 54. getDeploymentResult Code Sample in SupplyCraftAgent Class.

```
/**
 * Update the voters' active decision based on the previous result
 * obtained.
 * Decisions == Prev stands Decisions != Prev stands
 * ------------------------------------------------------
 *      IMPROVEMENT   DEPROVEMENT   IMPROVEMENT  DEPROVEMENT
 *
 * GO      +1           -1           -1           +1
 *
 * STAY    +1           -1           -1           +1
 * Only the winner will have the previous stand as TRUE.
 */
private void updateVoters(boolean improvement) {
   // Update all Scoring basing on previous Deployment result.
   // Need winner (who is winner), improvement(winner result)
   // Who is the previous 2nd winner, whom will be the only one with
   // GO.
   int previousWinner = winner.get(winner.size()-2);
   // Update all the decision score including active decision.
   // For each SSC, traverse the voters.
   for (int SSCcount = 0; SSCcount < totalSSC; SSCcount ++) {
      int wasDeployed = 0;
      if (SSCcount == previousWinner)
         wasDeployed = 1;
     // for each Voters, update the result.
     for (int voteCount = 0; voteCount < totalVoters; voteCount++) {
      // Current voter.
      Voter thisVote = voters[SSCcount][voteCount];
      if (improvement) {
         // 1st Decision
         if (thisVote.getDecisionList(0) == wasDeployed)
            thisVote.setDecisionScore(0,
                  thisVote.getDecisionScore(0) + 1);
         else
            thisVote.setDecisionScore(0,
                  thisVote.getDecisionScore(0) - 1);
         // 2nd Decision
         if (thisVote.getDecisionList(1) == wasDeployed)
            thisVote.setDecisionScore(1,
                  thisVote.getDecisionScore(1) + 1);
         else
            thisVote.setDecisionScore(1,
                  thisVote.getDecisionScore(1) - 1);
         // 3rd Decision
      if (thisVote.getDecisionList(2) == wasDeployed)
         thisVote.setDecisionScore(2,
            thisVote.getDecisionScore(2) + 1);
      else
         thisVote.setDecisionScore(2,
            thisVote.getDecisionScore(2) - 1);
      }
      else {
         // 1st Decision
         if (thisVote.getDecisionList(0) == wasDeployed)
            thisVote.setDecisionScore(0,
                  thisVote.getDecisionScore(0) - 1);
         else
            thisVote.setDecisionScore(0,
                  thisVote.getDecisionScore(0) + 1);
         // 2nd Decision
         if (thisVote.getDecisionList(1) == wasDeployed)
            thisVote.setDecisionScore(1,
                  thisVote.getDecisionScore(1) - 1);
```

```
         else
            thisVote.setDecisionScore(1,
                  thisVote.getDecisionScore(1) + 1);
         // 3rd Decision
         if (thisVote.getDecisionList(2) == wasDeployed)
            thisVote.setDecisionScore(2,
                  thisVote.getDecisionScore(2) - 1);
         else
            thisVote.setDecisionScore(2,
                  thisVote.getDecisionScore(2) + 1);
      }

      // Make the highest Scored decision the active Decision.
      int presentActive = thisVote.getActiveDecision();
      int presentActiveScore = thisVote.getDecisionScore(presentActive);
      int highestActive = presentActive;
      int highestScore = presentActiveScore;
      for (int decisionCount = 0; decisionCount < 3; decisionCount++) {
         if (thisVote.getDecisionScore(decisionCount) > highestScore) {
            highestScore = thisVote.getDecisionScore(decisionCount);
            highestActive = decisionCount;
         }
      }
      // Highest Score became the active Decision.
      thisVote.setActiveDecision(highestActive);
     }
   }
}
```

Figure 55. updateVoters Code Sample in SupplyCraftAgent Class.

LIST OF REFERENCES

[1] M. Bumble and L. Coraor, "Implementing Parallelism in Random Discrete Event-Driven Simulation," The Pennsylvania State University, 1998.

[2] E. Ros, R. Carrillo, E.M. Ortigosa, B. Barbour, and R. Agis, "Event-driven simulation scheme for spiking neural networks using lookup tables to characterize neuronal dynamics," Neural Computation, 18(12), pp. 2959-2993, December 1, 2006.

[3] M.L. Huson, "An Empirical Development of Parallelization Guidelines for Time-Driven Simulation," M. S. thesis, Defence Technical Information Center, December 1989.

[4] S. Cho, "A distributed time-driven simulation method for enabling real-time manufacturing shop floor control," in *Computers and Industrial Engineering*, vol. 49 issue 4, pp. 572-590, 2005.

[5] L.W. Schruben, *Graphical Simulation Modelling and Analysis: Using SIGMA for Windows*. The Scientific Press Series, 1995.

[6] A. Buss, "Component-Based Simulation Modeling," in *Proceedings of the 1996 Winter Simulation Conference*, pp. 964-971, 2000.

[7] A. Buss, "Component-Based Simulation Modeling with Simkit," in *Proceedings of the 2002 Winter Simulation Conference*, pp. 243-249, 2002.

[8] A. Buss, "Discrete Event Programming with Simkit," in *Technical Notes of Simulation News Europe*, issue 32/33, pp. 15-25, November 2001.

[9] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, "The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations," June 21, 1996.

[10] B. Dubiel and O. Tsimhoni, "Integrating Agent Based Modeling into Discrete Event Simulation," in *Proceedings of the 2005 Winter Simulation Conference*, pp. 1029-1037, 2005.

[11] G. Wagner, "The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior," in *Information Systems 28:5*, 2003.

[12] P. Riley, "Spades: A System for parallel-agent, discrete-event simulation," in *AI Magazine*, issue 24, vol 2, pp. 41-42, Summer 2003.

[13] A.M. Law, *Simulation Modeling & Analysis,* 4th Edition, McGraw-Hill, 2007.

[14] J.W. Schmidt and R.E. Taylor, *Simulation and Analysis of Industrial Systems*, Homewood, Illinois: Irwin, 1970.

[15] A. Buss, "Structure of DES Model," class notes for OA3302, Operations Research Department, Naval Postgraduate School at Monterey-California, Winter 2007.

[16] P. Davidsson, "Multi Agent Based Simulation: Beyond Social Simulation," Department of Software Engineering and Computer Science, University of Karlskrona/Ronneby Soft Center, Ronneby-Sweden, 2000.

[17] W.M. Bulleit and M.W. Drewek, "Simulating Terrorism in a Community," in *Proceeding of the Agent 2006 on Social Agents*, pp. 255-263, 2006.

[18] M.J. North, C.M. Macal and J.R. Vos, "Terrorist Organization Modeling," in *Proceedings of NAACSOS Conference*, 2004.

[19] J. Hiles, "MAS Design Template," class notes for MV4015, Modeling, Virtual Environments and Simulation (MOVES) Institute, Naval Postgraduate School at Monterey-California, Spring 2007.

[20] R.G. Sargent, "Event Graph Modelling for Simulation with An Application to Flexible Manufacturing Systems," in *Management Science*, vol. 34, issue 10, pp. 1231-1251, October 1988.

[21] T.K. Som and R.G. Sargent, "A Formal Development of Event Graphs as an Aid to Structured and Efficient Simulation Programs," in *ORSA J. Computing, 1*, pp. 107-125, 1989.

[22] A. Buss, "Event Graph Models and Simkit", class notes for OA3302, Operations Research Department, Naval Postgraduate School at Monterey-California, Winter 2007.

[23] Swarm Development Group, "Swarm Development Group Wiki," Internet: http://www.swarm.org/wiki/Main_Page#Swarm_Development_Group, 26 February 2007 [accessed 7 December 2007].

[24] G. Wagner, "The Agent-Object-Relationship Meta-Model: Towards a Unified View of State and Behavior," in *Information Systems*, issue 28, vol 5, pp. 475-504, 2003.

[25] P. Riley (pfr+@cs.cmu.edu), "SPADES," Project post to SourceForge.Net, 2007, Internet: http://sourceforge.net/project/showfiles.php?group_id=61929 [accessed 7 December 2007].

[26] A. Buss, "Basic Event Graph Modeling," in *Simulation News Europe*, issue 31, pp. 1-6, April 2001.

[27] W.B. Arthur, "Inductive Reasoning and Bounded Rationality," in *Papers and Proceedings of American Economic Review*, issue 84, pp. 406-411, 1994.

[28] M.F. Galli, et. al., "Riverine Sustainment 2012," M.S. thesis, Naval Postgraduate School, Monterey, CA, U.S.A., June 2007.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Arnold H. Buss
   Naval Postgraduate School
   Monterey, California

4. Professor Christian J. Darken
   Naval Postgraduate School
   Monterey, California

5. Professor John Hiles
   Naval Postgraduate School
   Monterey, California

6. Professor Rudolph P. Darken
   Naval Postgraduate School
   Monterey, California

7. Professor Mathias Kolsch
   Naval Postgraduate School
   Monterey, California

8. Professor Curtis Blais
   Naval Postgraduate School
   Monterey, California